Nonblocking On-Chip Interconnection Networks

Dissertation

Vom Fachbereich Informatik der Technischen Universität Kaiserslautern zur Verleihung des akademischen Grades Doktor der Ingenieurwissenschaften (Dr.-Ing.) genehmigte Dissertation

von

Tripti Jain

Datum der wissenschaftlichen Aussprache — 26.07.2019

Dekan — Prof. Dr. Stefan Deßloch

Gutachter — Prof. Dr. Klaus Schneider

Prof. Dr. Onur Mutlu

D 386

Abstract

Interconnection networks enable fast data communication between components of a digital system. The selection of an appropriate interconnection network and its architecture plays an important role in the development process of the system. The selection of a bad network architecture may significantly delay the communication between components and decrease the overall system performance.

There are various interconnection networks available. Most of them are blocking networks. Blocking means that even though a pair of source and target components may be free, a connection between them might still not be possible due to limited capabilities of the network. Moreover, routing algorithms of blocking networks have to avoid deadlocks and livelocks, which typically does only allow poor real-time guarantees for delivering a message.

Nonblocking networks can always manage all requests that are coming from their input components and can therefore deliver all messages in guaranteed time, i.e, with strong real-time guarantees. However, only a few networks are nonblocking and easy to implement. The simplest one is the *crossbar* network which is a comparably simple circuit with also a simple routing algorithm. However, while its circuit depth of $O(\log(n))$ is optimal, its size increases with $O(n^2)$ and quickly becomes infeasible for large networks. Therefore, the construction of nonblocking networks with a quasipolynomial size $O(n \log(n)^a)$ and polylogarithmic depth $O(\log(n)^b)$ turned out as a research problem. Beneš [Clos53; Bene65] networks were the first non-blocking networks having an optimal size of $O(n \log(n))$ and an optimal depth of $O(\log(n))$, but their routing algorithms are quite complicated and require circuits of depth $O(\log(n)^2)$ [NaSa82].

Other nonblocking interconnection networks are derived from *sorting networks*. Essentially, there are merge-based (MBS) and radix-based (RBS) sorting networks. MBS and RBS networks can be both implemented in a pipelined fashion which leads to a big advantage for their circuit implementation. While these networks are nonblocking and can implement all $n!$ permutations, they cannot directly handle partial permutations that frequently occur in practice since not every input component communicates at every point of time with an output component. For merge-based sorting networks, there is a well-known general solution called the Batcher-Banyan network. However, for the larger

class of radix-based sorting networks this does not work, and there is only one solution known for a particular permutation network.

In this thesis, new nonblocking radix-based interconnection networks are presented. In particular, for a certain permutation network, three routing algorithms are developed and their circuit implementations are evaluated concerning their size, depth, and power consumption. A special extension of these networks allows them to route also partial permutations. Moreover, three general constructions to convert any binary sorter into a ternary split module were presented which is the key to construct a radix-based interconnection network that can cope with partial permutations. The thesis compares also chip designs of these networks with other radix-based sorting networks as well as with the Batcher-Banyan networks as competitors. As a result, it turns out that the proposed radix-based networks are superior and could form the basis of larger manycore architectures.

ACKNOWLEDGEMENT

First and foremost, I want to express my deep gratitude to my advisor Prof. Klaus Schneider for his meticulous care, kindness and generosity. He is an excellent teacher and a great motivator. These four years, which I have spent under his guidance have brought me to a level where I can attempt to explore further the sea of knowledge on my own. I will always be indebted to him for all the guidance and assistance. I always find myself captivated by the simplified and intuitive approach that he follows to explain complicated concepts. I am extremely thankful for the tremendous time and effort that he invested in correcting my papers and thesis. It is not just guidance in teaching or research, I found him helping me in all aspects of personal development. He always aspires me to become a better person. He has always taken time out of his busy schedule to listen to my problems, however small they are, and I will always be indebted for that. I appreciate all his contributions of time, ideas, and funding to make my Ph.D. experience productive and stimulating. The joy and enthusiasm he has for his research was contagious and motivational for me, even during tough times in the Ph.D. pursuit.

I am also thankful to Marita Stuppy who has always helped me during my stay at TU Kaiserslautern. Her timely help has ensured the smooth execution of all the administrative tasks. I am also thankful to all of my colleagues which during the course of my stay here became my very good friends. It would have been impossible to work without having those fun moments which we shared together. I would like to specially thank Xiao, Maximilian, Omair, Mark, Martin and Anoop for all those technical and non-technical discussions which we had from time to time. I am also thankful to my friends Vivek, Sunita, Neelima, Rahul, Sagar, and Atul with whom I enjoyed my best moments in life. They made me feel home during my stay in Germany. I wish to express heartfelt love and appreciation to my dear friend and my fiance Ankesh for his constant inspiration, support and encouragement. Throughout this journey, many more people have provided their support and encouragement and I am thankful to each one of them.

At last but not the least, I would like to dedicate this thesis to my parents Mr. Paras Jain and Mrs. Mona Jain who have always been a source of inspiration, infinite love and affection. I owe this completely to them and without their care and support, I could not have come this far. I would also

like to thank my uncles, my sisters and brother-in-laws for their unconditional support and care.

December 2018, Tripti Jain

Contents

1. Introduction **1**
- 1.1. Motivation . . . 1
- 1.2. Contributions . . . 3
- 1.3. Outline . . . 5

2. State of the Art **7**
- 2.1. Basics of Interconnection Networks . . . 7
- 2.2. Multistage Interconnection Networks . . . 14
- 2.3. Sorting Networks . . . 20
- 2.4. Current Commercial Interconnection Networks . . . 23
- 2.5. Summary . . . 24

3. New RBS Networks for Total Permutations **25**
- 3.1. Basic Definitions: . . . 26
- 3.2. Distribution-Based Interconnection Networks . . . 28
- 3.3. Sorters with Half Cleaner based RBS Network . . . 40
- 3.4. Summary . . . 44

4. New RBS networks for Partial Permutations **47**
- 4.1. Routing Partial Permutations by Sorting Networks . . . 47
- 4.2. Routing Partial Permutations . . . 49
- 4.3. Summary . . . 54

5. Experimental Results and Comparison **57**
- 5.1. Asymptotic Complexities . . . 57
- 5.2. Experimental Results . . . 58
- 5.3. Analysis of Results . . . 60

6. Conclusions **63**

Bibliography **65**

A. Experimental Tables **73**

B. Experimental Graphs **87**

C. Curriculum Vitae **101**

List of Figures

2.1. Bus architecture 11
2.2. 2-D mesh architecture 11
2.3. 2-D Torus architecture 12
2.4. Ring architecture 12
2.5. Binary tree architecture 13
2.6. Hypercube architecture 13
2.7. Crossbar architecture 14
2.8. The two states of a 2×2 crossbar switch. 15
2.9. Examples of 8×8 bidelta networks 16
2.10. Dilated multistage interconnection network $k = 2$ 17
2.11. 8×8 replicated multistage network $k = 2$ 18
2.12. 4×4 fat tree network 18
2.13. Clos network 19
2.14. Beneš network 19
2.15. Flattened bufferfly network 20
2.16. Merge-based sorting network 21
2.17. Radix-based sorting network 21
2.18. An 8-input Narasimha's concentrator 22
2.19. An 8-input Koppelman and Oruç's concentrator 22
2.20. Chien-Oruç's merge circuit 23

3.1. Recursive construction of binary sorter for n inputs/outputs. . 28
3.2. Entire Interconnection Network based on Radix-based Sorting. 29
3.3. Circuits for $n = 2$ and $n = 4$ for Parallel Prefix Computation. . . 31
3.4. Implementing *Split* modules by two sorting networks and a half cleaner module. 40
3.5. Compare-and-swap switch (the arrow points towards the minimum output). 41
3.6. A half cleaner module with 8 inputs and 8 outputs consisting of four 2×2 compare-and-swap switches (the arrows of compare-and-swap switches point towards the minimum output). 41

3.7. Possible/impossible inputs and outputs of compare-and-swap switches of the half cleaner under the assumptions given in Lemma 3.2: The first and the last input/output rows cannot occur, and therefore $y_i \neq 1$ and $y_{i+n} \neq 0$ holds for $i = 0, \ldots, n-1$. 43
3.8. A *RBS* network for 8 inputs constructed by *Split* modules according to Figure 3.4. 44

4.1. Converting Sequences to Prefix Sequences. 50
4.2. Entire Interconnection Network based on Radix-based Sorting. 50
4.3. Ternary RBS network. 51
4.4. Construction of a Ternary Sorter by two Binary Sorters. 52
4.5. Construction of a Ternary Splitter by Binary Sorters. 54
4.6. Construction of a Ternary Splitter by Ternary Sorters and a Half Cleaner. 55

B.1. CNC-BIN-HC0 88
B.2. CNC-BIN-HC1 89
B.3. CNC-TRP-HC0 90
B.4. CNC-TRP-HC1 91
B.5. CNC-TRC-HC0 92
B.6. NET-BIN-HC0 93
B.7. NET-BIN-HC1 94
B.8. NET-TRP-HC0 95
B.9. NET-TRP-HC1 96
B.10. NET-TRC-HC0 97
B.11. TNT-BIN-HC0 98
B.12. Comparison with Others Networks 99

List of Tables

2.1. Commercial multicore architectures and their interconnection networks 24

3.1. Configuration of switch i depending on the parity $p_i := x_0 \oplus \ldots \oplus x_{2i}$ of the inputs. 29
3.2. Ranking-based configuration p_i of switch i in the network shown in Figure 3.1 based on the most significant bits $\mathsf{msb}(x_{2i})$,$\mathsf{msb}(x_{2i+1})$ of the target addresses and the ranks ranks r_{2i}, r_{2i+1} of the inputs x_{2i}, x_{2i+1}, respectively. We also determine the local addresses/ranks rL_i and rU_i to be used in the lower and upper subnetworks for further routing. 35

5.1. Comparsion of asymptotic complexities 58

A.1. CNC-BIN-HC0 74
A.2. CNC-BIN-HC1 75
A.3. CNC-TRP-HC0 76
A.4. CNC-TRP-HC1 77
A.5. CNC-TRC-HC0 78
A.6. NET-BIN-HC0 79
A.7. NET-BIN-HC1 80
A.8. NET-TRP-HC0 81
A.9. NET-TRP-HC1 82
A.10. NET-TRC-HC0 83
A.11. TNT-BIN-HC0 84
A.12. Comparison With Other Networks 85

Chapter 1

Introduction

1.1. Motivation

The rapid advancement of technology and process scaling has helped in realizing computers clocked at several GHz. However, process scaling has already approached an atomic level and the researchers have started discussing the end of Moore's law [PaAv12]. It is now very difficult to further increase the performance of computational devices based on synchronous circuits by increasing the clock frequency. This motivates the designer to explore other possible architectures which are either not running on a clock, i.e., asynchronous circuits, or circuits which do not use a single global clock. The design of asynchronous circuits does not have the limitations of clock skew and electromagnetic interference (EMI). However, ensuring a glitch-free operation in case of asynchronous designs is very challenging and EDA tools are not fully supportive for these designs. Therefore, instead of using fully asynchronous circuits, a better strategy is to use locally synchronous circuits having globally asynchronous interconnections also classified as GALS (globally asynchronous and locally synchronous) architectures in the literature where the individual local synchronous circuits can also use different clocks. This will allow one to optimize the power consumption and will also help in reducing the EMI effects.

Another dimension to further enhance the performance of computers is to exploit parallelism like pipelining in its architectures. Among many potential ways to still increase the performance of future chip designs, one that already became successful lead to system-on-a-chip designs where entire systems are connected on a single chip. This has lead to the development of multicore processor architectures. Compared to a single core architecture where the core interacts with the memory and the input output systems through a bus, a multicore architecture has several cores and a common bus interface to interact with the external memory and peripheral input and output systems. Essentially, all new processor architectures are now multicore architectures with an increasing number of cores.

Intel's SCC (single-chip cloud computer) and Xeon Phi architectures as well as the modern programmable GPUs are one of the examples of the upcom-

ing manycore architectures that integrate a large number of cores on a single chip. Other examples are Raw [LBFS98] with the commercial variant Tilera [BEAC08], WaveScalar [SSMP07], TRIPS [BKMD04], Flexcore [TSBS07], explicit datapath wide SIMD [WSCH15], SCAD [BhJS15] and the Transport-Triggered architectures (TTAs) [Corp94]. These architectures provide a large number of processing units and the compiler is not only responsible to schedule the instructions to these processing units but also to move data from one processing unit to another. Hence, systems-on-a-chip designs are nowadays prevalent and require efficient interconnection networks for the communication of the single systems on the chip. De Micheli and also others [BeMi02; Mein03] therefore already predicted the need for networks on a chip to cope with the communication bandwidths needed in these designs.

The increasing number of on-chip components imposes new challenges for developing efficient interconnection networks that make a good compromise between the chip size, latency, complexity of routing algorithms, and the bandwidth of the networks. There are many different interconnection networks that can be considered for this purpose (see [BjMa06; Feng81; AgIS09; DaTo04] as surveys). So far, buses, ring structures, torus networks, and n-dimensional mesh networks have been considered in on-chip network research. However, the performance of these networks is limited since these networks are blocking networks which means that even though a pair of source and target components may be free, a connection between them might still not be possible due to limited capabilities of these networks. Moreover, routing algorithms have to avoid deadlocks, livelocks and starvation in these networks which only allows pessimistic estimations of real-time guarantees.

Nonblocking networks, on the other hand, are able to implement all $n!$ permutations of n components. Many nonblocking networks are already known: The simplest one is the *crossbar* network which is a comparably simple circuit with also a simple routing algorithm. However, while its circuit depth of $O(\log(n))$ is optimal, its size increases with $O(n^2)$ and quickly becomes infeasible for large manycore architectures. For this reason, Clos and Beneš [Clos53; Bene65] networks have been developed that are also nonblocking. Beneš networks have an optimal size of $O(n \log(n))$ and an optimal depth of $O(\log(n))$, but their routing algorithms are quite complicated and require circuits of depth $O(\log(n)^2)$ [NaSa82].

Other nonblocking interconnection networks are derived from *sorting networks*. Essentially, there are merge-based (MBS) (Figure 2.16) and radix-based (RBS) (Figure 2.17) sorting networks. MBS networks first split the given input list into halves, sort them independently and then merge the sorted halves into a single sorted list. RBS networks partition the given inputs, e.g., into two partitions such that all inputs in a partition are already in the right halves so that they can be recursively sorted.

The implementation of RBS networks mainly depends on the implementation of *Split* modules and the implementation of MBS networks depends on the implementation of $Merge$ modules. Their size and depth determines the size and depth of the entire network. Batcher has presented circuits for

Merge modules that lead to his famous bitonic sorter and oddeven sorter [Batc68]. Both have a size of $O(n \log(n)^2)$ and a depth of $O(\log(n)^2)$ in terms of compare-and-swap modules.

Split modules are usually implemented by binary sorters where the inputs are just sorted by their most significant bits of the target addresses. Many implementations of binary sorters are known [Nara94; KoOr90; ChCh96; ChOr94] that lead to circuits of RBS networks of size of $O(n \log(n)^3)$ and a depth of $O(\log(n)^2)$ or $O(\log(n)^3)$ in terms of constant fan-in/out circuit gates.

MBS and RBS networks can be both implemented in a pipelined fashion which leads to a big advantage for their circuit implementation compared to Beneš or Clos networks. However, there is another problem that has to be solved in practice: While these networks are nonblocking and can implement all $n!$ permutations, they cannot directly handle partial permutations that frequently occur in practice since not every input component communicates at every point of time with an output component. However, only two solutions are known that extend sorting networks to nonblocking interconnection networks that can deal with partial permutations: (1) The Batcher-Banyan network extends Batcher's networks by a further back-end permutation network [HuKn84; Nara88] and (2) Narasimha showed that his RBS network can deal with partial permutations when extended by a front-end permutation network [Nara94].

Hence, the overall problem considered in this thesis is the circuit design of nonblocking interconnection networks that can handle both total and partial permutations.

1.2. Contributions

In order to implement *Split* and *Merge* modules for RBS and MBS interconnection networks, this work first focuses on permutation networks. While some permutation networks can be used as *Split* and *Merge* modules, others cannot be used as such. A systematic methodology to analyze permutation networks by means of binary decision diagrams (BDDs) has been presented in [JaSc16]. The proposed method can be applied to all permutation networks. This work verified that some of the permutation networks can be used as *Split* and *Merge* modules provided that a specific permutation of the outputs is added to the permutation network.

In particular, the permutation network used by Narasimha's concentrator belongs to the interesting class of powerful permutation networks. As Narasimha already showed, it leads to a RBS network that can handle partial permutations. However, this concentrator has a bad circuit depth of $O(n)$ which is impractical. The first contribution of the thesis is therefore a parallel implementation [JaSJ17] of Narasimha's concentrator that only has a depth of $O(\log(n)^2)$ while keeping its size in $O(n \log(n)^2)$.

In [JaSJ17], details of the implementation of the new *Split* modules are described, and the asymptotic complexity of the circuit size and depth are

presented and compared with other interconnection networks. Moreover, that network as well as other related RBS networks have been implemented using 65nm CMOS chip technology. The maximal frequency, the required chip area, and the power consumption of these circuits were compared. The evaluation shows that the new network is best among the considered radix-based networks regardless whether the maximal frequency, the chip area or the power consumption was considered.

Another contribution of the thesis shows a special variant of Koppelman and Oruc's concentrator that can, in contrast to the original version, also deal with partial permutations. In [JaSc18c], details of the implementation of the *Split* modules are described, and the asymptotic complexity of the circuit size and depth are presented and compared with other interconnection networks.

Splitters can be implemented either using concentrators or binary sorters. It is well-known that the use of more general concentrators instead of binary sorters may lead to more efficient circuits. However, the design of concentrators turned out to be a challenging task for many decades. In [JaSJ17b], we proposed that one can construct from any binary sorter a corresponding concentrator by means of Batcher's half cleaner circuit. This way, one can improve any existing *Split* module that is implemented by a sorter.

Special modifications are required for sorting networks for routing partial permutations. For merge-based sorting networks, there is a well known solution known as the Batcher-Banyan network. However, for the larger class of RBS networks this does not work, and there is only one solution known by Narasimha's network that can route partial permutations.

One way to route partial permutations in RBS networks is to replace the binary *Split* modules by ternary *Split* modules. General ways to convert any binary sorter into a ternary sorter or directly to a ternary *Split* module are presented in [JaSc18a; JaSc18c]. These circuits almost maintain the depth of the binary sorters, but essentially double their size. The half-cleaner optimization also works for ternary sorters [JaSc18; JaSJ17c] and may therefore almost halven the size again. This generalization is based on the use of Batcher's half cleaner circuit that has the ability to partition sorted sequences according to their most significant bits.

Moreover, in [JaSc18c] we proved that the improved configuration logic developed in [JaSJ17] for the reverse banyan flip-shuffle network also used by Narasimha does also route partial permutations provided a front-end concentrator is added. Together with the half cleaner optimization, this leads to a very efficient RBS network that can route all partial permutations.

To summarize, this thesis proposes three new non-blocking unicast interconnection networks based on the radix sorting scheme. It also successfully analyses different RBS networks and proves the efficacy of the proposed interconnection networks through analytic and simulation results. We have further proposed a scheme to design interconnection networks with partial permutations using ternary *Split* modules.

1.3. Outline

This work presents new non-blocking interconnection networks that can handle both total and partial permutations. It is organized as follows: In Chapter 2, we discuss the theoretical background of the contributions presented in this thesis, covering a wide range of topics that are used throughout this thesis. This chapter is focused on the necessary technical background to understand the rest of the thesis.

In Chapter 3, we first discuss constructions of new non-blocking unicast interconnection networks based on the radix sorting scheme for total permutations. This also includes the basic definitions of the terms used to describe these new RBS networks. Then, we discus algorithms and asymptotic complexities of these new networks.

In Chapter 4, we first prove that the particular class of configuration circuits can be extended to route partial permutations. Furthermore, we also discuss alternatives for implementing partial permutations by implementing ternary *Split* modules

In Chapter 5, we first report about the asymptotic complexity obtained for proposed networks and compare it with other networks. Then, we report the comparison of experimental results obtained by implementations of various binary sorters and networks. Our experimental results were made by a netlist generator written in F# and the Cadence® RC compiler (version 14.2) using 65nm CMOS technology.

In Chapter 6, we summarize presented work with some conclusions.

Chapter 2

State of the Art

Contents

2.1. Basics of Interconnection Networks 7
2.1.1. Basic Terminology 8
2.1.2. Network Aspects 9
2.1.3. Network Architectures 10
2.2. Multistage Interconnection Networks 14
2.2.1. Permutation Networks 14
2.2.2. Dilated Multistage Network 17
2.2.3. Replicated Multistage Network 17
2.2.4. Fat Tree Networks 18
2.2.5. Clos Networks 18
2.2.6. Beneš Networks 19
2.2.7. Flattened and 2-Dilated Flattened Butterfly Networks 20
2.3. Sorting Networks 20
2.3.1. Merge-based Sorting Networks 20
2.3.2. Radix–based Sorting Networks 21
2.4. Current Commercial Interconnection Networks . . 23
2.5. Summary 24

2.1. Basics of Interconnection Networks

In general, interconnection networks have the task to connect a fixed number n of input components with n output components. At any point of time, each one of the input components $I_0, \ldots, I_{n-1}$ may provide an input pair (m_i, t_i) where m_i is a message for the output component O_{t_i} with target address $t_i \in \{0, \ldots, n-1\}$. The task of the interconnection network is to deliver the message m_i of input component I_i as efficient as possible to output component O_{t_i} whose address is t_i. The efficiency of an interconnection network usually refers to the following parameters,

- **Latency**: It defines the time required to deliver an individual message from the input to the output of the network.
- **Throughput or bandwidth**: It defines the capacity of a network to transmit an amount of data from the input to the output in a unit interval of time usually, expressed in bits per second [bits/sec].
- **Hardware cost**: It refers to the cost of the implementation of the interconnection network.

An ideal network supports a small size, a high bandwidth and a low latency, even though there exists a tradeoff between these parameters. The bandwidths become important when trying to minimize the hardware cost of the interconnection network. The message size, the length of a message in bits, is another important design consideration, affect the performance of the interconnection network. The message size can be reduced by dividing them into smaller units, called packets. This thesis considers only latency, throughput and hardware cost parameters.

2.1.1. Basic Terminology

Terminal Node and Switch Node

A node that acts as a source and sink for the data is called a terminal node. On the other hand, a node that forwards data from input ports to output ports is called a switch node.

Direct and Indirect Networks

In a direct network, every node is both a switch node and a terminal node. Messages are transferred from a source node to a destination node via intermediate nodes. Some well-known examples of a direct networks are ring, star, and mesh networks. In an indirect network, a node is either a terminal node or a switch node. The messages are transferred from a source node to a destination node via switch nodes. Some well-known examples of an indirect networks are crossbar switches [ScRe39], multistage networks like Ω -permutation networks [Lawr75], Banyan networks [OrOr85],[GoLi98], fat trees [Leis85a], flattened butterfly networks [KiDA07], 2-dilated butterfly networks [ThCh10], and Beneš networks [Clos53],[Bene64],[Bene65],[Bene75],[Waks69].

Static and Dynamic Networks

In a static network, the connections between source and destination nodes are fixed and can't be changed. The examples of this type of network are rings, stars,chordal rings, etc. In contrast, the connections of a dynamic network can be reconfigured. Examples of dynamic networks are multistage networks like the Ω -permutation network, Beneš, clos etc.

Blocking and Nonblocking Networks

Nonblocking networks allow every input node to be connected to any output node that is not also the target of another input node. Mathematically speaking, such networks can implement all permutations to map their inputs to their outputs. A crossbar network [ScRe39] is an example of a non-blocking network which allows all components to communicate with each other at the same point of time. In contrast, blocking networks can't implement all permutations to map their inputs to their outputs. Banyan networks [OrOr85],[WuFe80] are examples of blocking networks.

Multicast and Unicast Networks

In a multicast network, single input components may send a message to several output components. In contrast, A unicast network corresponds with one to one mappings where one input component may send a message to exactly one output component.Both multicast and unicast networks may have target address conflicts.

Self-Routing Networks

A network that does not need any additional setup time to route the data, since the target address directly defines the conflict free routes, is called a self-routing network. Delta networks and sorting networks are examples of self-routing networks.

2.1.2. Network Aspects

There are four common aspects that must be considered in the design of any interconnection network. These aspects determine how the interconnection network is implemented and also affect the cost and performance of the network.

Topology

The topology of an interconnection network specifies the way connections are wired. Basically, it defines how nodes and links are connected. Since the topology dictates the total number of alternate paths a message can take to reach the destination, it directly affects the bandwidth and latency of the network. The network reliability is also greatly influenced by the topology.

Routing Algorithms

Routing algorithms determine the specific path a message will take from a source to a destination for a given network topology. If many routes are possible, the goal of the routing algorithm is to distribute traffic evenly among the paths supplied by the network topology and to minimize contention, thus improving network latency and throughput.

Flow Control

The flow control assigns different resources such as buffers and channels as the message progresses from source to destination. Basically, a flow control protocol dictates how the message actually traverses the assigned route, including when a message must be buffered and when it leaves a intermediate node through the desired outgoing link. A good flow control protocol can lower the latency and increase the network throughput. Proper flow control is also required to avoid deadlocsk and livelocks in the network.

Microarchitecture

The microarchitecture defines how the router is organized. Basically, it realizes the routing and flow control protocols and critically shape the circuit implementation.

2.1.3. Network Architectures

The most important and critical step is to choose the network architecture for a communication system. The network architecture includes the network topology, the buffer sizes, the buffer positions, and so on. The chosen architecture must fulfill all the requirements of the expected network traffic.

Network architectures are classified as wired or wireless. In this thesis, we focused on wired network architectures. Many wired network architectures have been proposed [BjMa06; Feng81; AgIS09; DaTo04; ScJu96]. Some of them are briefly discussed below.

Bus

A bus (Figure 2.1) provides a simple broadcast medium and an easy-to-use communication schema where all terminals are connected via the same bus with each other. The source terminal initiates the communication by allocating the bus. It transmits the target address and the message via the bus. All terminals listen to the bus and compare this target address with their own. An address match identifies the target terminal which reads the message. Finally, the bus is deallocated.

Due to the concept of a single common bus, only two nodes can communicate with each other at a time. Thus, the bandwidth becomes a major bottleneck as the number of nodes to be connected increases. Also, the power usage per communication event grows as more units are added, since the further attached units lead to higher capacitive loads [BjMa06].

Mesh Networks

For networks on chip where the processing elements are arranged in a matrix, mesh networks are quite popular. They connect the input/output terminals via routers and switches to the grid network. It can be one dimensional, two

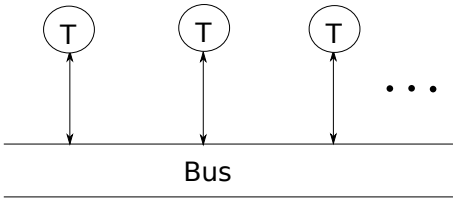


Figure 2.1.: *Bus architecture*

dimensional, or three dimensional. Figure 2.2 shows a two dimensional mesh network.

Each router is connected to its two nearest neighbors in each dimension. Four bidirectional links handle all the communications between the routers of a 2-D mesh. The number of links per router does not change if additional terminals are added to the mesh network. Therefore, a mesh network offers very good scalability.

The communication is usually implemented by sending messages from one router to another one where the routers have to determine the route to one of its neighbors that will then be responsible for the further routing. A mesh network consumes however a substantial part of the overall energy and also claims a large part of the chip size [BJSH15; SDMS12]. Moreover, due to the local decisions of the routers, it is essentially impossible to guarantee bounds on the time required for a message to arrive at its target address [JiYa14; KeME16]. Depending on the used routing algorithms [Hols09; HeWC04], it may even lead to livelocks and deadlocks. However, there are methods to avoid deadlocks like x-y routing etc.

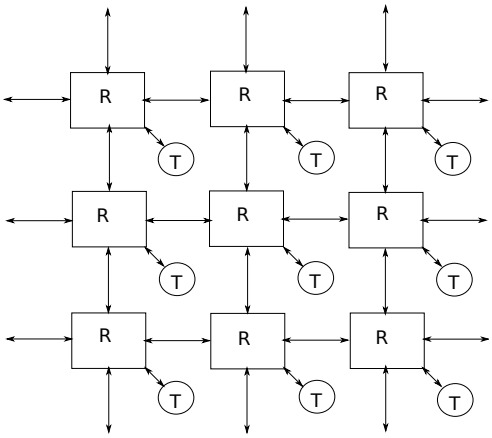


Figure 2.2.: *2-D mesh architecture*

Torus Networks

The torus is an extension of the mesh architecture. It is a mesh architecture where all edge routers have additional links to their corresponding edge router at the opposite edge. Figure 2.3 depicts a two-dimensional torus (2-D torus).

A torus is a direct network, like a mesh. Due to the similar structure, a torus reveals the same advantages and drawbacks as a mesh. However, a mesh architecture is not symmetric on the edges, so the Torus avoids this problem with the price of a more difficult chip layout.

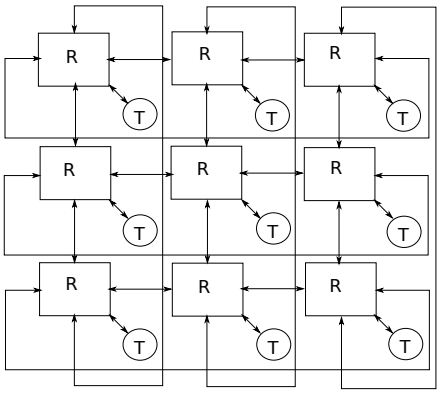


Figure 2.3.: *2-D Torus architecture*

Ring Networks

The ring is another static/direct architecture. In such an architecture, each router is connected to exactly two other routers, one on each side, leading to an overall structure of a closed loop (Figure 2.4). The target address and message are sent to the ring, and usually circle in a common direction from router to router.

The main drawbacks of the ring architecture are: (a) low reliability (entire network is affected if any link fails), (b) high latency, and (c) not easy to scale.

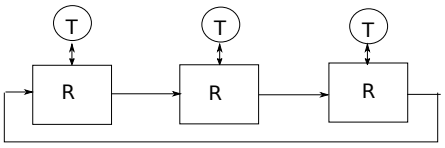


Figure 2.4.: *Ring architecture*

Tree Networks

A Tree architecture is an indirect network. In the tree architecture, all routers are arranged as a tree. A root router is connected to two successor routers and these routers are again connected to their successor routers, and so on. The

routers with no further successor routers are called leafs and are the terminal nodes. If all the routers are connected to a fixed number k of successors (except the leafs or terminals), the network architecture is called a k-ary tree. Figure 2.5 shows a binary tree which is also called balanced because all terminal nodes have the same distance to the root node. In such an architecture, the communication between any node in the left half with any node in the right half is always established via the root router. Therefore, the root router acts as a bottleneck of the network. An alternative structure named fat tree [Leis85a] which overcomes this problem is described in Sect. 2.2.

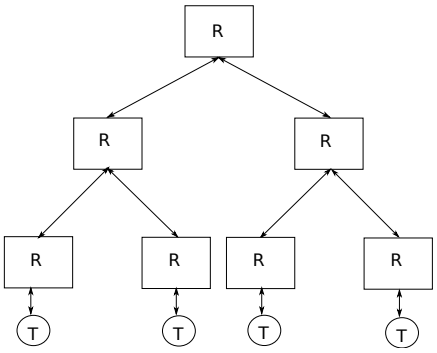


Figure 2.5.: *Binary tree architecture*

Hypercube Networks

The hypercube [RJMC95] is another direct network used frequently in parallel computing. The nodes of the hypercube also represent the nodes of the network. Figure 2.6 shows a three-dimensional hypercube.

Concerning the blocking behavior, equal problems arise as with a mesh network: It provides low latency and its chip layouts are quite difficult.

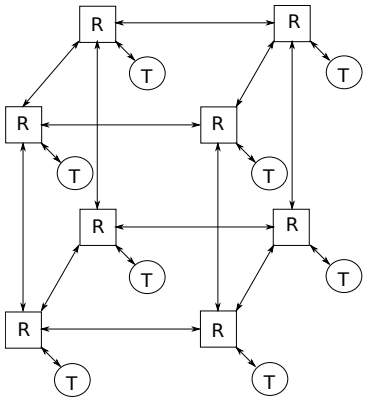


Figure 2.6.: *Hypercube architecture*

Crossbar Networks

Crossbars [ScRe39; BjMa06] are dynamic networks consisting of a switch matrix. It is very simple to design and allows all n components to communicate with each other at the same point of the time (i.e., it is non-blocking). Figure 2.7 shows a 4×4 crossbar, consists of four inputs and four outputs. The switches are located at the crosspoints of the horizontal and vertical lines. Each switch corresponds to a specific input-output pair connect the related source and destination terminals.

Since n inputs and n outputs produce n^2 crosspoints, it means crossbars are poorly scalable: its size increases with $O(n^2)$; the main drawback of the crossbars. Hierarchically connected crossbars, e.g., multistage interconnection networks (Sect.2.2), avoid this drawback.

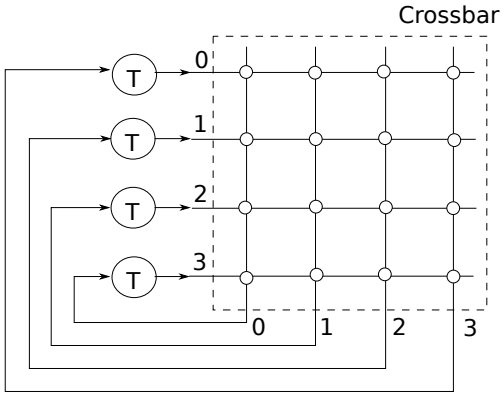


Figure 2.7.: *Crossbar architecture*

2.2. Multistage Interconnection Networks

To overcome the drawbacks of crossbars, multistage interconnection networks (MINs) [Lawr75; Lee85; RaVa87; NgDu01; Cam03] have been introduced. MINs are indirect/dynamic networks based on switching elements (SEs). The switching elements are constructed by $k \times k$ crossbar switches (for some small numbers of k like 2 or 4), and are arranged in stages and connected by interstage links.

The interstage link structure and number of switching elements characterize multistage interconnection networks. Several multistage interconnection network architectures exist. Some of them are briefly discussed below.

2.2.1. Permutation Networks

Permutation networks are typically built of 2×2 crossbar switches that are organized for a $n \times n$ permutation network in a grid with $\frac{n}{2}$ rows and $\log_2(n)$ columns. Such a 2×2 crossbar switch with inputs x_0 and x_1 and outputs y_0 and y_1 can be brought in one of two states (see Figure 2.8) controlled by a

further select input s that is typically not drawn. If the select input s is 0, it is in 'through' mode, thus mapping its inputs x_0 and x_1 to its outputs y_0 and y_1, respectively, and if the select input s is 1, it is in 'crossed' mode, thus mapping its inputs x_0 and x_1 to its outputs y_0 and y_1, respectively.

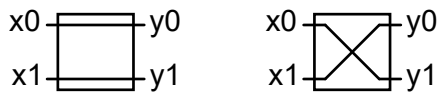


Figure 2.8.: *The two states of a 2×2 crossbar switch.*

Permutation networks for the same number of inputs and the same number of switches may differ in many aspects, in particular, on how many and which permutations they can implement, and on how the configurations can be computed for a desired permutation.

The delta network is an $a^n \times b^n$ switching network with n input stages, consisting of $a \times b$ crossbar modules. This network has one and only one path from any input to any output and thus covers a very large class of possible network structures [KrSn83; KrSn86]. Rectangular delta networks additionally demand square sized switching elements (i.e., equal number of switching element inputs and outputs). In practice, an $a^n \times b^n$ crossbar module for a delta network is more cost-effective if a and b are powers of 2. Bidelta networks (i.e., delta property for input-to-output direction and output-to-input direction), include the Omega, Flip, Baseline, Indirect Binary Cube (IBC), modified data manipulator and reverse baseline networks which have all been proven topologically equivalent in [WuFe80]. Examples of 8×8 networks are shown in Figure 2.9. Their interstage connections distinguish them.

As can be seen, the topology of the bidelta network depends on two parameters: First, use a specific permutation between the columns of the network, and second, recursively partition the columns into blocks where these permutations are applied. If no partitioning is applied, i.e., the selected permutation is applied in every column to all of the n values, we call it a multistage network, if the number of blocks is doubled from right to left, we call it a *banyan* network, and if the number of blocks is doubled from left to right, we call it a *reverse banyan* network.

Some permutations (interstage connections) used for permutation networks are defined below in terms of the binary representation $a_{p-1}, \ldots, a_0$ of an address where $p = \log_2(n)$ bits are required for n inputs/outputs:

- $PerfectShuffle(a_{p-1}, \ldots, a_0) := (a_{p-2}, \ldots, a_0, a_{p-1})$
- $FlipShuffle(a_{p-1}, \ldots, a_0) := (a_0, a_{p-1}, \ldots, a_1)$
- $Butterfly(a_{p-1}, \ldots, a_0) := (a_0, a_{p-2}, \ldots, a_1, a_{p-1})$

As can be seen, the perfect shuffle permutation rotates the leftmost bit to the right, and the flip shuffle permutation rotates the rightmost bit to the left. Hence, both permutations are the inverses of each other. The butterfly

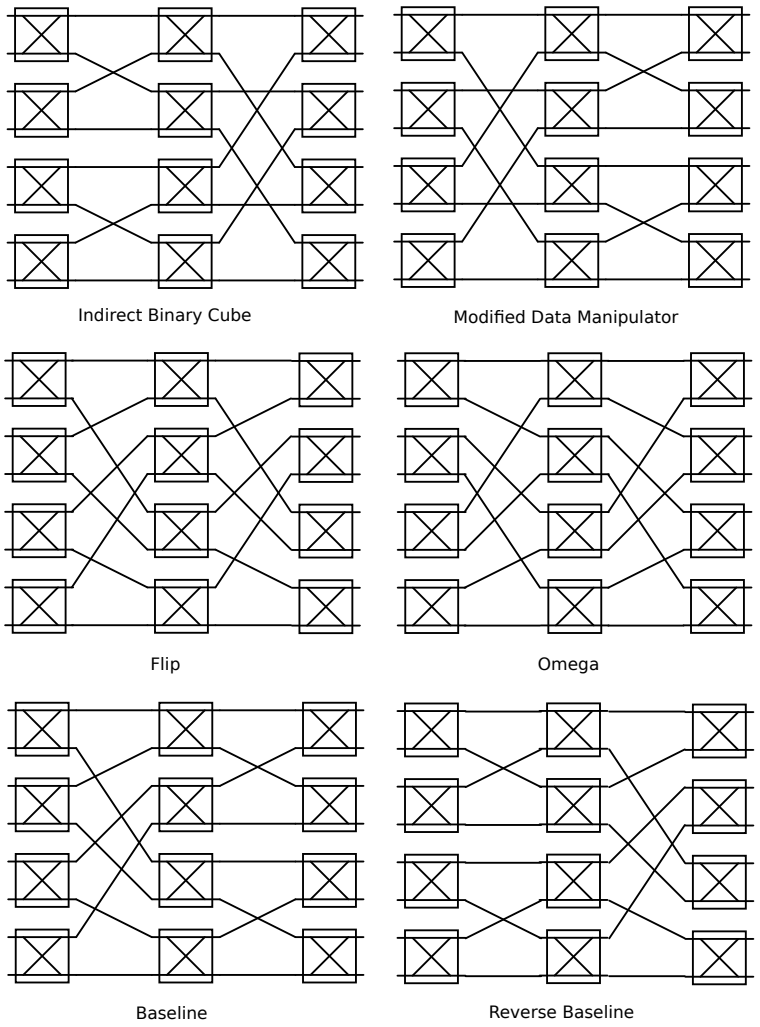


Figure 2.9.: *Examples of* 8×8 *bidelta networks*

permutation exchanges the leftmost and rightmost bits and is therefore inverse to itself. All the bidelta networks can be renamed, on the bases of the rules of the interstage connections and topology, are given in Table 2.1

Network	*Topology*	*Permutation*
$IndirectBinaryCube := BanyanButterfly$	$Banyan$	$Butterfly$
$ModifiedDataManipulator := ReverseBanyanButterfly$	$ReverseBanyan$	$Butterfly$
$Flip := MultistageFlipShuffle$	$Multistage$	$FlipShuffle$
$Omega := MultistagePerfectShuffle$	$Multistage$	$PerfectShuffle$
$Baseline := ReverseBanyanFlipShuffle$	$ReverseBanyan$	$FlipShuffle$
$ReverseBaseline := BanyanFlipShuffle$	$Banyan$	$FlipShuffle$

The general delta network is formally proven to be self-routing in [Pate81]. On the other hand, it is also well-known that these networks are necessarily *blocking* i.e., they cannot implement all $n!$ permutations of the inputs.

2.2.2. Dilated Multistage Network

The concept of dilated multistage networks was introduced by Kruskal and Snir [KrSn83]. This network reduces blocking by replicating the interstage connection lines k times. Then, the crossbar switch size must be increased by a factor of k to ensure the required number of inputs and outputs for the interstage connection lines. Figure 2.10 shows the architecture of an 8×8 dilated multistage network with all interstage connection lines doubled ($k = 2$). It allows transmitting up to k packets from a particular crossbar column stage i to at stage $i + 1$. Blocking may occur if more than k packets are sent.

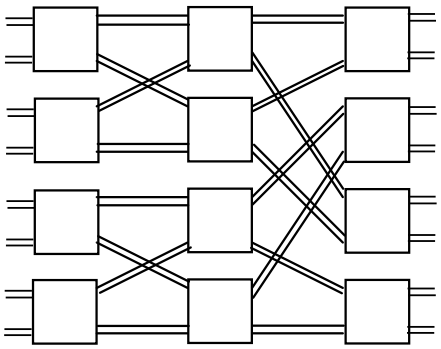


Figure 2.10.: *Dilated multistage interconnection network $k = 2$*

2.2.3. Replicated Multistage Network

Similar to the dilated, replicated multistage networks were also introduced by Kruskal and Snir [KrSn83]. This network also reduces blocking by replicating the network k times. Figure 2.11 shows the architecture of an 8×8 replicated multistage network consisting of 2×2 crossbar switches and two layers.

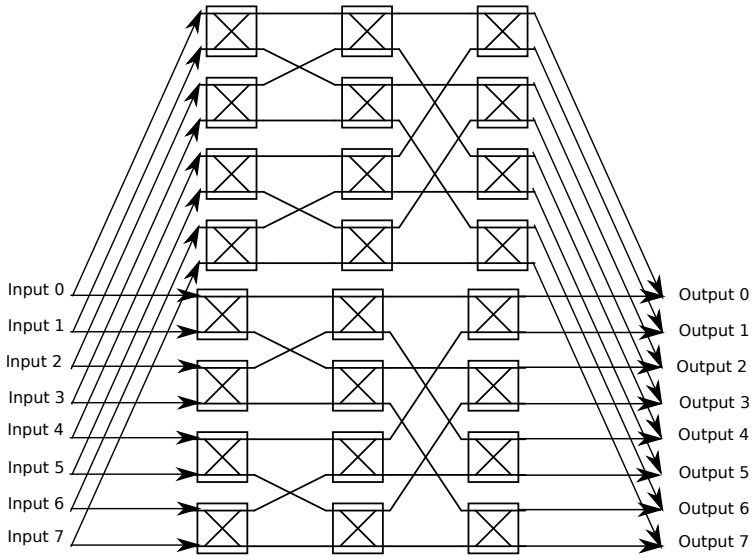


Figure 2.11.: 8×8 *replicated multistage network* $k = 2$

2.2.4. Fat Tree Networks

Generally, tree based networks work with a simple routing, and the distance between two nodes is no more than $2\log(n)$ for a tree with n leaves. In a binary tree, links at higher levels of the tree have to potentially deal with more traffic than those at the lower levels. Therefore, the root router acts as a bottleneck of the network. Fat tree networks [Leis85a] overcome this problem by doubling the number of links at every level of the tree. As the number of communication links increases from the leaf nodes to the root, the communication bandwidth increases as well. Figure 2.12 shows a 4×4 fat tree network. Fat binary trees are nonblocking networks.

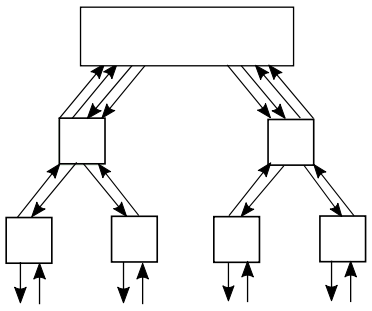


Figure 2.12.: 4×4 *fat tree network*

2.2.5. Clos Networks

Clos networks [Clos53; TuMe03] are nonblocking multistage networks. The basic Clos network consists of three stages. A symmetric Clos network is

characterized by a triple, (m, n, r) where m is the number of middle-stage switches, n is the number of input (output) ports on each input (output) switch, and r is the number of input and output switches [DaTo04], as shown in the Figure 2.13. Clos networks may be recursively decomposed into a five stage, a seven stage and so on by replacing each switch in the central stage by a three stage Clos network. Clos has shown in [Clos53] that his network is strictly non-blocking if the condition $m \geq (2n - 1)$ holds.

Clos networks provide path diversity, but their routing is difficult and also has a cost that is nearly twice of that of the Banyan butterfly.

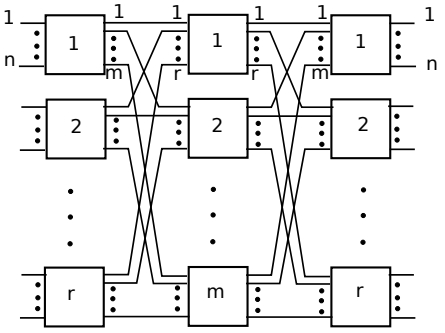


Figure 2.13.: *Clos network*

2.2.6. Beneš Networks

Beneš networks [Clos53; Bene64; Bene65; Bene75; Waks69; Kann05] are also nonblocking networks [Pipp78a], built by extending a baseline network with its inverse one. The last stage of the baseline and the first stage of the inverse baseline are shared. The Beneš network consists of $\frac{n}{2}$ rows and $2\log(n) - 1$ columns of 2×2 crossbar switches. Figure 2.14 shows the Beneš network of size 8×8.

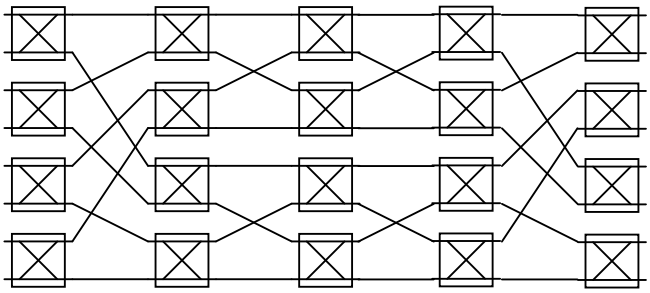


Figure 2.14.: *Beneš network*

It offers enough path diversity to implement all possible $n!$ permutations of mapping its n inputs to its n outputs. Setting up the switches of the Beneš network to implement a desired permutation requires however a non-trivial routing algorithm [Andr77; NaSa81; NaSa82; FeSe94; LeLi96; KiYM97; SeFL99;

CaFo99; LuZh02] which requires at least $O(n \cdot \log(n))$ time with sequential algorithms [NaSa82]. Beneš networks can also be viewed as recursively constructed by Clos networks.

2.2.7. Flattened and 2-Dilated Flattened Butterfly Networks

Flattened butterfly [KiDA07] is built by combining or flattening the routers in each row of the network into a single router of a conventional butterfly network. Examples of flattened butterfly constructions are shown in Figure 2.15.The flattened butterfly is a blocking network, and this blocking behavior can degrade the performance of the network [ThCh10].

A 2-dilated flattened butterfly [ThCh10] is derived from a flattened butterfly. This network reduces blocking by duplicating the interstage connection lines of the flattened butterfly.

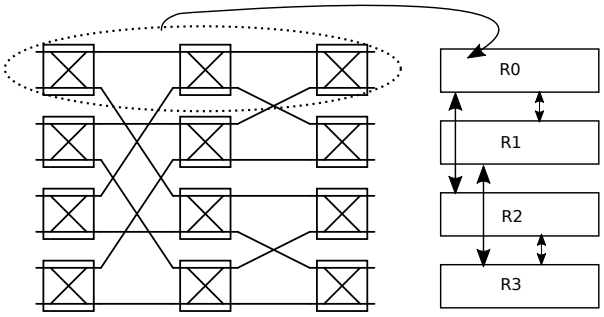


Figure 2.15.: *Flattened bufferfly network*

2.3. Sorting Networks

Sorting networks are another way to implement interconnection networks where the input messages are sorted with respect to their target addresses. Sorting networks are non-blocking networks. Sorting networks are built by 2×2 compare-and-swap modules that compare the target addresses of two inputs and swaps them if needed to generate the two outputs. This way, these networks are self-routing, i.e., do not require additional configuration logic. There are two important classes of such a sorting networks, namely the merge-based (MBS) and the radix-based (RBS) sorting networks.

2.3.1. Merge-based Sorting Networks

The merge-based sorting paradigm is shown in Figure 2.16. In the merge-based approach, a sorting network MBS (n) for n inputs is recursively constructed by splitting the given sequence into two halves, recursively sorting these by sorting networks MBS $(\frac{n}{2})$ of half the size, and then merging the sorted halves by a merge module $Merge(n)$. The well known and the best practical merge

based MBS networks are Batcher's bitonic and oddeven sorters having a size of $O(n \log n^2)$ and a depth of $O(\log n^2)$ in terms of compare-swap modules.

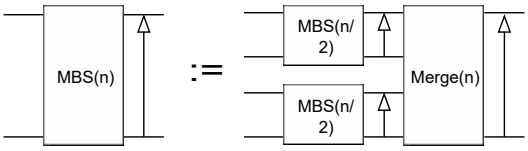


Figure 2.16.: *Merge-based sorting network*

2.3.2. Radix–based Sorting Networks

The radix-based sorting paradigm is shown in Figure 2.17. In radix-based sorting networks, the given inputs are partitioned into two halves by a $Split(n)$ module according to the most significant bit of their target address. Thus, after the $Split(n)$ module, the given inputs have already been routed to the right halves, so that the remaining problems can be dealt with recursively in the same way (ignoring now the most significant bits of the target addresses).

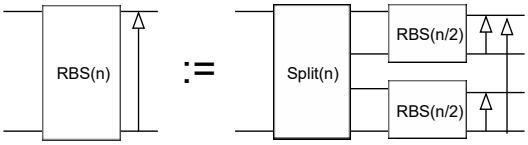


Figure 2.17.: *Radix-based sorting network*

The implementation of radix-based sorting networks mainly depends on the implementation of the $Split(n)$ modules. Their size and depth determines the size and depth of the entire network. There are many ways to implement a $Split(n)$ module, e.g., by means of concentrators [Pins73; Pipp77; Chun78; MaGN79]: A (n, m)-concentrator is a circuit with n inputs and $m \leq n$ outputs that can route any given number $k \leq m$ of valid inputs to k of its m outputs. Split modules can therefore be implemented by two $(n, \frac{n}{2})$-concentrators: One that routes the $\frac{n}{2}$ inputs with a most significant target address bit 1 from the n inputs to the $\frac{n}{2}$ outputs, and another one routing the other $\frac{n}{2}$ inputs with a most significant target address bit 0 to its outputs. The two concentrators' outputs are then the upper and lower halves of the Split module's outputs.

Most of the practically useful $Split(n)$ modules are implemented by permutation networks. Some of them are briefly discussed below.

Narashima's Networks

Basically, Narasimha's networks are a recursive construction of a binary sorter based on the reverse banyan flip shuffle network. This binary sorter routes the valid inputs to the output array with the highest indices.

Narasimha's binary sorter is shown in Figure 2.18. The aim of the leftmost column, consisting of 2×2 switches, is to distribute the 0s and 1s equally in

two halves that are then recursively treated in the same way. A final output permutation converts the generated sequence into a sorted one. The routing algorithm is easy to implement. However, the performance is quite slow due to its large depth of $O(n)$.

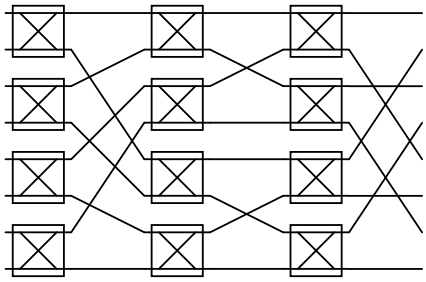


Figure 2.18.: *An 8-input Narasimha's concentrator*

Koppelman and Oruç's Networks

Koppelman and Oruç's networks [KoOr90] are also a recursive construction of a binary sorter based on the cube network. As shown in Figure 2.19, the routing of the permutation network is based on a ranking circuit. The ranking circuit computes local addresses of all valid inputs which are then used as target addresses to route the inputs through the permutation network.

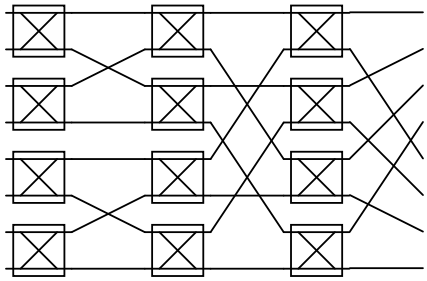


Figure 2.19.: *An 8-input Koppelman and Oruç's concentrator*

Chien and Oruç's Networks

Chien and Oruç's networks [ChOr94] are based on the parallel merge-sort paradigm. Chien and Oruc's merge circuit for n inputs is shown in Figure 2.20. The merge algorithm described in [ChOr94] is based on so-called bi-sorted binary sequences, i.e., sequences whose two halves are sorted sequences. The key observation for their binary sorter is that if a sorted binary sequence is cut into two halves, then one of the two halves either contains only 0s or only 1s (called a clean sequence) while the other one is still a sorted sequence. This observation is used in a recursive construction where multiplexers are used to

select the unclean half for recursive treatment that is then concatenated with the clean half. Their binary sorter has a depth of $O(\log(n)^2)$ and a size of $O(n\log(n)^2)$.

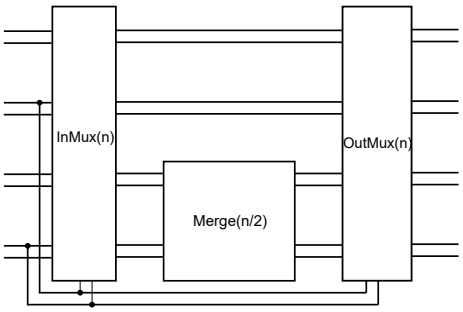


Figure 2.20.: *Chien-Oruç's merge circuit*

Cheng and Chen's Networks

Cheng and Chen's [ChCh96] networks are another radix-based sorting interconnection networks based on the generalized cube permutation network. The idea of that network is based on the consideration of so-called compact sequences which are sequences of the form $0^i1^j0^k$ or $1^i0^j1^k$. This sophisticated circuit computes the configuration of a permutation network and yields a binary sorter with a depth of $O(\log(n))$ and a size of $O(n\log(n))$.

2.4. Current Commercial Interconnection Networks

Recently, there has been a large number of multicore architectures designed and manufactured for the commercial market which are developed by industry as well as research groups. These multicore architectures target a range of applications covering embedded, general purpose desktop, and server realms. Therefore, the selection of an appropriate interconnection network and its architecture to connect processor cores plays an important role in the development process of the system. The selection of a bad network architecture may significantly delay the communication between the processing cores and decrease the system performance.

As already mentioned in this chapter, there are various interconnection networks available for intercore communications. Each one of them has its own pros and cons in terms of scalablity, simplicity, and performance. All traditional interconnection networks are realized using a bus structure. However, with an increasing number of cores, the bus structure does no longer meets the needs of new multicore architectures. Therefore, buses have been replaced by mesh networks, but also mesh networks have limited capabilities.

Table 2.1 presents different commercial multicore architectures with interconnection networks. Some of them are general purpose multicores, some are specialized for specific application domains and some are more specialized to target high performance in their application domains.

Multicore Architectures	Number of Cores	Interconnect Networks
Intel Larrabee	up to 48	Bidirectional ring
Microsoft Xenon	three	Crossbar
Intel Core I7	2 to 8	Point to point
Sun Niagara T2	8	Crossbar
Intel Atom	1 to 2	Bus
ARM Cortex- A9	1 to 4	Bus
Element CXI Eca-64	4 clusters of 1 core + alu	Hierarchial noc
Tilera Tile64	36 to 64	NoC
Intel ASCI Red Pragon	2 with 4510 number of nodes	2D mesh
IBM ASCI White SP Power3	16 with 512 number of nodes	Bidirectional MIN with 8-ports bidirectional switches (typically a fat tree or Omega)
Intel Thunder Itanium2 Tiger4	4 with 1024 number of nodes	Fat tree with 8-port bidirectional switches
Cray XT3	1 with 30,508 number of nodes	3D torus
Cray X1E	1 with 1024 number of nodes	4-way bristled 2D torus with express links
IBM ASC Purple pSeries 575	8 with more than 1280 number of nodes	Bidirectional MIN with 8-ports bidirectional switches (typically a fat tree or Omega)
IBM Blue Gene/L eServer Solution	2 with 65,536 number of nodes	3D torus
MIT Raw	16 network ports (16 tiles or cores)	2D mesh
IBM Power5	7 network ports (2 PE cores and 5 other ports)	Crossbar
U.T. Austin TRIP Edge	40 network ports (16 L2 tiles and 24 network interface tile)	2D mesh
Sony, IBM,Toshiba Cell BE	12 network ports	Ring

Table 2.1.: *Commercial multicore architectures and their interconnection networks*

2.5. Summary

Most of the interconnection networks like buses, torus, n-dimensional meshes etc, are blocking networks. The performance of these networks is limited since even though a pair of source and target components may be free, a connection between them might still not be possible due to limited capabilities of these networks. Non-blocking networks offer ideal performance, but very few are non-blocking networks. Crossbars are one of them which are simple in design, but the size increases with $O(n^2)$. Beneš and Clos, are also nonblocking, have an optimal depth and size, but their routing algorithms are quite complicated. Other non-blocking interconnection networks are derived from sorting networks. Batcher's famous bitonic sorter and oddeven sorter have a size of $O(n\log(n)^2)$ and a depth of $O(\log(n)^2)$ in terms of compare-and-swap modules. $\mathsf{S}split(n)$ modules, the basic building block of radix-based sorting networks, are usually implemented by binary sorters where the inputs are just sorted by their most significant bits of the target addresses. Many implementations of binary sorters are known that lead to circuits of RBS networks of size of $O(n\log(n)^3)$ and a depth of $O(\log(n)^2)$ or $O(\log(n)^3)$ in terms of simple circuit gates.This thesis considers efficient circuits for implementation of non-blocking interconnection networks.

Chapter 3

New RBS Networks for Total Permutations

Contents

3.1. Basic Definitions: **26**
3.1.1. Total Permutations 26
3.1.2. Partial Permutations 26
3.1.3. Splitter 26
3.1.4. Concentrator 27
3.1.5. Ternary Sorter 27
3.2. Distribution-Based Interconnection Networks . . . **28**
3.2.1. Correctness of the Binary Sorter 29
3.2.2. Switch Configuration Circuits 31
3.3. Sorters with Half Cleaner based RBS Network . . **40**
3.3.1. The Half Cleaner Lemma 40
3.4. Summary **44**

Nonblocking unicast interconnection networks allow every input component to be connected to any output component that is not also the target of another input component. The efficient implementation of such networks turned out to be a difficult challenge for many decades. Sorting networks are attractive alternatives for the design of such networks.

Therefore, we present new nonblocking unicast interconnection networks based on radix sorting scheme. First, we discuss the basic definitions of the terms that we will use to describe these new RBS networks. Then, we describe the construction of these new RBS networks. Finally, we discuss the algorithms, and the asymptotic complexities of these new networks.

To analyze the complexity of networks for a number of inputs n, we consider the following well-known circuit complexity measures:

- **Depth:** The depth is the length of the longest path through combinational gates from inputs to outputs in the considered circuit.

- **Size:** The size is the number of gates[1] used in the circuit where only 2-input/1-output gates are allowed.

3.1. Basic Definitions:

The interconnection network can be described as a set of interconnection functions, where each is a permutation (bijection) on the set of port numbers which is either source or destination addresses [Lai00]. Addresses of input and output components are numbered as $0, \ldots, n-1$ in the following.

3.1.1. Total Permutations

Definition 3.1 ⟨ Total Permutations ⟩
Total permutations are all bijective mappings of $0, \ldots, n-1$ *to itself, i.e, uniquely mapping. Since all the inputs need the connection to one of the outputs, sorting networks can be used to implement total permutations.*

3.1.2. Partial Permutations

Definition 3.2 ⟨ Partial Permutations ⟩
Partial permutations are partially defined injective functions from $0, \ldots, n-1$ *to itself. Since some of the inputs may not need a connection to a target address, their target addresses are invalid, denoted as* $\perp$ *in this thesis.*

3.1.3. Splitter

Definition 3.3 ⟨ Splitter ⟩
A splitter is a module that permutes its $2n$ *inputs* $x_0, \ldots, x_{2n-1}$ *over values* $\{0, \perp, 1\}$ *to its* $2n$ *outputs* $y_0, \ldots, y_{2n-1}$ *such that all input sequences where at most* n *of the inputs* x_i *are* 0 *and at most* n *of the inputs* x_i *are* 1 *are mapped to outputs satisfying the following properties:*

- $y_i \in \{0, \perp\}$ *for* $i = 0, \ldots, n-1$
- $y_{n+i} \in \{\perp, 1\}$ *for* $i = 0, \ldots, n-1$

Input sequences where more than n *values are* 0 *or more than* n *inputs are* 1 *may be permuted to any output sequence*[2].

[1] It is well-known that another set of gates yields at 3-most a constant blow-up since all 2-input/1-output gates can be implemented with a constant number of other 2-input/1-output gates that form a boolean basis.

Hence, all $x_i = 0$ are routed to the lower half $y_0, \ldots, y_{n-1}$ and all $x_i = 1$ are routed to the upper half $y_n, \ldots, y_{2n-1}$. The halves do not have to be sorted, i.e., the elements $\{0, \perp\}$ in the lower half and the elements $\{\perp, 1\}$ in the upper half may appear in any order.

An important case are binary input sequences, i.e., sequences where no value is $\perp$. In that case, the condition mentioned in the definition means that exactly n values are 0 and the other n values are true, and the consequence demands that the output sequence is sorted. We do however not demand that other binary input sequences will be sorted, and therefore splitters are even in the case of binary sequences more general than sorters.

3.1.4. Concentrator

Definition 3.4 ⟨ Concentrator ⟩
A (n, m)-concentrator is a circuit with n inputs and $m \leq n$ outputs that can route any given number $k \leq m$ of valid inputs to k of its m outputs.

In the following, we are mainly interested in $(n, \frac{n}{2})$-concentrators. Considering inputs 1 as the valid ones, it is clear that we can implement a $(n, \frac{n}{2})$-concentrators by any splitter with n inputs simply by ignoring the lower half, and analogously, considering inputs 0 as the valid ones, we can implement a $(n, \frac{n}{2})$-concentrators by simply ignoring the upper half of outputs.

Conversely, we can construct a splitter by means of two concentrators: One concentrator considers input values 0 as valid ones and routes these $m \leq \frac{n}{2}$ input values 0 to its $\frac{n}{2}$ outputs together with $\frac{n}{2} - m$ values $\perp$. The other concentrator considers input values 1 as valid ones and routes these $m' \leq \frac{n}{2}$ input values 1 to its $\frac{n}{2}$ outputs together with $\frac{n}{2} - m'$ values $\perp$. Since (n, m)-concentrators have a minimal size of $O(n)$ [Pins73], we conclude that there are also splitters of size $O(n)$.

3.1.5. Ternary Sorter

Definition 3.5 ⟨ Ternary Sorter ⟩
A ternary sorter permutes its n inputs $x_0, \ldots, x_{n-1}$ over values $\{0, \perp, 1\}$ to a sorted output sequence $y_0, \ldots, y_{n-1}$, i.e., where $y_i \preceq y_{i+1}$, with the total order $0 \preceq \perp \preceq 1$.

Clearly, any ternary sorter with $2n$ inputs is also a splitter, since the input sequences where at most n inputs are 0 and 1, respectively, are sorted as any other input sequence, and therefore all values 0 are mapped to the lower half

and all values 1 are mapped to the upper half, respectively. However, ternary sorters also sort the other half, and also all other input sequences which are considered as don't care values for splitters.

As a consequence, ternary sorters are more specific, and therefore their minimal size is $O(n\log(n))$ [HoSR98] while splitters and concentrators can be built with size $O(n)$.

3.2. Distribution-Based Interconnection Networks

As any interconnection network based on radix-sorting (see the Chapter 2), also this new network mainly depends on the construction of a suitable $Split(n)$ module. The core idea of implementing the $Split(n)$ module is the use of a binary sorter similar to the one presented by Narasimha [Nara94]. This $Split(n)$ module, which is like Narasimha's binary sorter, is recursively constructed as shown in Figure 3.1. Both networks discussed in this section are based on this binary sorter.

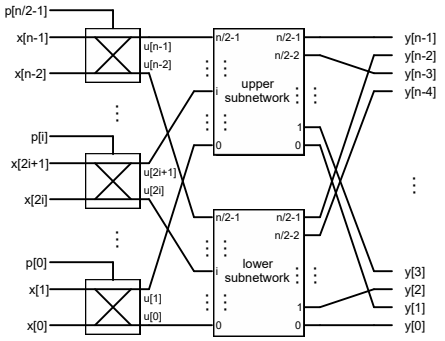


Figure 3.1.: *Recursive construction of binary sorter for n inputs/outputs.*

Figure 3.1 shows the construction of the binary sorter for n inputs $x_0, \ldots, x_{n-1}$. As can be seen, it consists of a column of $\frac{n}{2}$ 2×2 crossbar switches that are controlled by inputs $p_0, \ldots, p_{\frac{n}{2}-1}$. Switch i is thereby in one of two modes: If the configuration input p_i is 0, it is in 'through' mode, thus mapping its inputs x_{2i} and x_{2i+1} to its outputs u_{2i} and u_{2i+1}, respectively, and if the configuration p_i is 1, it is in 'crossed' mode, thus mapping its inputs x_{2i} and x_{2i+1} to its outputs u_{2i+1} and u_{2i}, respectively. Note that the outputs u_{2i} and u_{2i+1} of switch i are mapped to the lower and upper subnetwork's input i, respectively (where the lower and upper one's local addresses $0, \ldots, \frac{n}{2}-1$ are associated with the global addresses $0, \ldots, \frac{n}{2}-1$ and $\frac{n}{2}, \ldots, n-1$, respectively).

It can be directly seen in Figure 3.1 that the permutation between the columns of the switches is a flip-shuffle permutation fs that maps the address bits $a_{p-1}, \ldots, a_0$ to $a_0, a_{p-1}, \ldots, a_1$ and that the output permutation shown in Figure 3.1 is the perfect shuffle permutation ps that maps the address bits

p_i	x_{2i}	x_{2i+1}	u_{2i}	u_{2i+1}	remark
0	0	1	0	1	$up_1 = low_1$, send x_{2i+1} to upper
0	1	0	1	0	$up_1 = low_1 + 1$, send x_{2i} to lower
1	0	1	1	0	$up_1 = low_1 + 1$, send x_{2i+1} to lower
1	1	0	0	1	$up_1 = low_1$, send x_{2i} to upper
$*$	0	0	0	0	same inputs, same outputs
$*$	1	1	1	1	same inputs, same outputs

Table 3.1.: *Configuration of switch i depending on the parity $p_i := x_0 \oplus \ldots \oplus x_{2i}$ of the inputs.*

$a_{p-1}, \ldots, a_0$ to $a_{p-2}, \ldots, a_0, a_{p-1}$. At the end, we obtain[3] a reverse banyan network with flip-shuffle fs permutations between its stages, and a reverse-bit rv output permutation that maps address bits $a_{p-1}, \ldots, a_0$ to $a_0, \ldots, a_{p-1}$. A complete RBS-network based on this permutation network is shown in Figure 3.2, and we can see *Split* modules for $n = 8, 4, 2$ inputs, and all of them are constructed according to Figure 3.1.

We will prove in the next section that the switches should be configured such that the 1s (and 0s) of the binary input sequence $x_0, \ldots, x_{n-1}$ are equally distributed to the two subnetworks, and in case that the number of 1s (and 0s) is odd, then the additional 1 is routed to the upper subnetwork (and the additional 0 is routed to the lower subnetwork). For this reason, the final output permutation that routes outputs i of the lower and upper subnetworks to the final outputs y_{2i} and y_{2i+1}, respectively, will shuffle the sorted binary sequences coming from the subnetworks into a single binary sorted sequence.

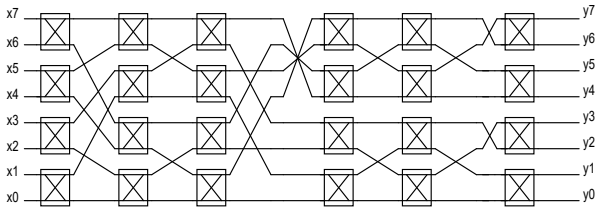


Figure 3.2.: *Entire Interconnection Network based on Radix-based Sorting.*

3.2.1. Correctness of the Binary Sorter

Here, we prove that the recursive implementation as given in the previous section implements a binary sorter. *The overall idea is to distribute the 1s (and therefore also the 0s) of any prefix of the inputs $x_0, \ldots, x_i$ equally to the two subnetworks, and in case the number of 1s is odd, we route the additional 1 to the upper and the additional 0 to the lower subnetwork.*

[3] This can be proved by induction where the induction step is proved as follows:

$$\begin{aligned} ps(a_{p-1}, rv(a_{p-2} \ldots, a_0)) &= ps(a_{p-1}, a_0, \ldots, a_{p-2}) \\ &= a_0, \ldots, a_{p-1} = rv(a_{p-1}, \ldots, a_0) \end{aligned}$$

This can be achieved by configuring the switches by the parities $p_i := x_0 \oplus \ldots \oplus x_{2i}$ of the inputs as explained below:

- If $x_{2i} = x_{2i+1}$ holds, the configuration of switch i does not matter since then the same number of 0s and 1s are routed from this switch to the two subnetworks in any case.

- If $x_{2i} \neq x_{2i+1}$ holds, one of the two is 0 while the other one is 1. Depending on the number of 1s that occurred in $x_0, \ldots, x_{2i-1}$, the new 1 has to be routed to the lower or upper subnetwork so that our equal distribution of 1s is maintained. To this end, assume that up_1 and low_1 denote the numbers of 1s occurring in $x_0, \ldots, x_{2i-1}$ that were already sent to the upper and lower subnetworks, respectively. Since we either have $up_1 = low_1$ or $up_1 = low_1 + 1$, depending on whether the number of 1s was even or odd, respectively, we just need to know whether the number of 1s in $x_0, \ldots, x_{2i-1}$ is even or odd. Now note that $p_i \oplus x_{2i} = x_0 \oplus \ldots \oplus x_{2i-1}$ holds (since $x \oplus x = 0$ and $x \oplus 0 = x$ holds in general). Hence, we have the following:

  - $up_1 = low_1$ if and only if $p_i \oplus x_{2i} = 0$ holds.
  - $up_1 = low_1 + 1$ if and only if $p_i \oplus x_{2i} = 1$ holds.

  Inspecting the cases shown in Table 3.1 finally shows that using $p_i := x_0 \oplus \ldots \oplus x_{2i}$ as configuration of switch i will equally distribute the 1s in $x_0, \ldots, x_{2i+1}$ to the two subnetworks.

Hence, with the computed configuration, the 1s contained in the input sequence $x_0, \ldots, x_{n-1}$ are equally distributed to the two subnetworks in case their number is even, and otherwise the additional 1 is sent to the upper and the additional 0 is sent to the lower subnetwork.

We finally prove the correctness of the binary sorter based on the above equal distribution property by induction on the number n of inputs/outputs:

induction base: For $n = 2$, the network consists of a single 2×2 crossbar switch. If we use for such a single crossbar switch $p_0 := x_0$ as its configuration, it can be easily seen by Table 3.1 that it will sort its two binary inputs. This already proves the induction base.

induction step: In the induction step, we may assume by the induction hypothesis that the two subnetworks in Figure 3.1 implement binary sorters. Now, it is important that we have equally distributed the 1s of the input sequence $x_0, \ldots, x_{n-1}$ to the two subnetworks, and in case their number was odd, the additional 1 is in the upper network. Because of this, the final perfect shuffle permutation will produce a sorted sequence of the two sorted subsequences of the local subnetworks (note that the sorted subsequences are either both $0^i 1^{\frac{n}{2}-i}$ or $0^{i+1} 1^{\frac{n}{2}-(i+1)}$ from the lower and $0^i 1^{\frac{n}{2}-i}$ from the upper subnetwork).

3.2.2. Switch Configuration Circuits

In the previous section, we have seen that the configuration of switch i should be computed as $p_i := x_0 \oplus \ldots \oplus x_{2i}$. The simplest circuit to implement this configuration logic has been suggested by Narasimha in [Nara94]. Narasimha's configuration logic leads to a circuit size of $O(n \log(n))$, and depth $O(n)$ which is quite slow. Therefore, we presented two circuits to efficiently compute the switch configurations $p_i = x_0 \oplus \ldots \oplus x_{2i}$ as required by previous section.

Parallel Prefix Computation

For computing the configuration of the switches, we have to compute all prefixes $\mathcal{P}_i := a_0 \oplus \ldots \oplus a_i$ for $i = 0, \ldots, n-1$ of a binary sequence $a_0, \ldots, a_{n-1}$. This can be done with the following well-known work-efficient parallel algorithm that we only explain for the case where n is a power of 2 (it works however also for any number n):

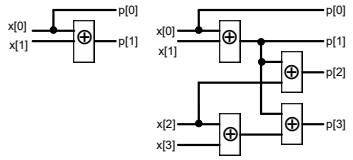


Figure 3.3.: *Circuits for $n = 2$ and $n = 4$ for Parallel Prefix Computation.*

1. Compute the following list using $\frac{n}{2}$ XOR gates in one step: $[b_0, \ldots, b_{\frac{n}{2}-1}] = [a_0 \oplus a_1, a_2 \oplus a_3, \ldots, a_{n-2} \oplus a_{n-1}]$, i.e., we define $b_i := a_{2i} \oplus a_{2i+1}$ for $i = 0, \ldots, \frac{n}{2} - 1$.
2. Recursively apply the parallel prefix computation to the list $[b_0, \ldots, b_{\frac{n}{2}-1}]$ and obtain its prefixes $[c_0, \ldots, c_{\frac{n}{2}-1}]$. Thus, we have $c_j := b_0 \oplus \ldots \oplus b_j$ for $j = 0, \ldots, \frac{n}{2} - 1$.
3. Since $b_j := a_{2j} \oplus a_{2j+1}$ holds, we therefore have $c_j := a_0 \oplus a_1 \oplus \ldots \oplus a_{2j} \oplus a_{2j+1}$, and have therefore already the prefixes $\mathcal{P}_{2j+1} := c_j$ for the odd indices $2j+1$ for $j = 0, \ldots, \frac{n}{2} - 1$. The prefixes of the even indices $\mathcal{P}_{2j}$ are now computed in one parallel step as $\mathcal{P}_0 := a_0$ and $\mathcal{P}_{2j} := \mathcal{P}_{2j-1} \oplus a_{2j} = c_{j-1} \oplus a_{2j}$ for $j = 1, \ldots, \frac{n}{2} - 1$ using $\frac{n}{2} - 1$ XOR gates.

The correctness of the above algorithm is easily shown by induction where we use the base cases as shown in Figure 3.3. Hence, we obtain for the depth $D_{prefix}(n)$ and size $S_{prefix}(n)$ the following recursive formulas:

- $D_{prefix}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ 1 & : \text{for } n = 2 \\ 2 & : \text{for } n = 4 \\ D_{prefix}\left(\frac{n}{2}\right) + 2 & : \text{for } n > 4 \end{cases}$
- $S_{prefix}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ S_{prefix}\left(\frac{n}{2}\right) + n - 1 & : \text{for } n > 1 \end{cases}$

It can be easily proved by induction on n that we have the following solutions of the above recursive definitions:

- $D_{prefix}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ 1 & : \text{for } n = 2 \\ 2\log(n) - 2 & : \text{for } n > 2 \end{cases}$
- $S_{prefix}(n) = 2n - \log(n) - 2$ for all n that are powers of 2

Complexity Analysis of the Binary Sorter:
For computing the configuration of the switches in Figure 3.1, we have to compute the parities $p_i := x_0 \oplus \ldots \oplus x_{2i}$ for $i = 0, \ldots, \frac{n}{2} - 1$. To this end, we first compute the following values z_i for $i = 0, \ldots, \frac{n}{2} - 1$:

$$z_i := \begin{cases} x_0 & : \text{for } i = 0 \\ x_{2i-1} \oplus x_{2i} & : \text{for } i = 1, \ldots, \frac{n}{2} - 1 \end{cases}$$

The computation of all z_i can be done in one step using exactly $\frac{n}{2} - 1$ XOR gates. After this, we apply the parallel prefix computation of the previous section to the $\frac{n}{2}$ values $z_0, \ldots, z_{\frac{n}{2}-1}$. The result values $p_0, \ldots, p_{\frac{n}{2}-1}$ are then our values needed to configure the switches because of the following:

$$p_i := z_0 \oplus z_1 \oplus \ldots \oplus z_i = x_0 \oplus (x_1 \oplus x_2) \ldots \oplus (x_{2i-1} \oplus x_{2i})$$

Hence, we can compute all p_i for $i \in \{0, \ldots, \frac{n}{2} - 1\}$ of one column with a circuit having the following depth $D_{colCF}(n)$ and size $S_{colCF}(n)$ for $n > 1$:

- $D_{colCF}(n) = \begin{cases} 0 & : \text{for } n \leq 2 \\ D_{prefix}\left(\frac{n}{2}\right) + 1 & : \text{for } n > 2 \end{cases}$
- $S_{colCF}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ S_{prefix}\left(\frac{n}{2}\right) + \frac{n}{2} - 1 & : \text{for } n > 1 \end{cases}$

which means the following for all n that are powers of 2:

- $D_{colCF}(n) = \begin{cases} 0 & : \text{for } n \leq 2 \\ 2 & : \text{for } n = 4 \\ 2\log(n) - 3 & : \text{for } n > 4 \end{cases}$
- $S_{colCF}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ \frac{3}{2}n - \log(n) - 2 & : \text{for } n > 1 \end{cases}$

To compute the depth $D_{nodeCF}(n)$ and size $S_{nodeCF}(n)$ of the entire configuration logic for the binary sorter shown in Figure 3.1, we set up the following recursive formulas:

- $D_{nodeCF}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ D_{colCF}(n) + D_{nodeCF}\left(\frac{n}{2}\right) & : \text{for } n > 1 \end{cases}$
- $S_{nodeCF}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ S_{colCF}(n) + 2S_{nodeCF}\left(\frac{n}{2}\right) & : \text{for } n > 1 \end{cases}$

which have the following solutions:

- $D_{nodeCF}(n) = \begin{cases} 0 & : \text{for } n \leq 2 \\ \log(\frac{n}{2})^2 + 1 & : \text{for } n > 2 \end{cases}$
- $S_{nodeCF}(n) = \left(\frac{3}{2}n + 1\right)\log(n) - 4n + 4 \text{ for } n \geq 1$

In addition to the above size and depth for the configuration logic, we also have to add the switches needed in the binary sorter. It is easily seen that the recursive definition according to Figure 3.1 will generate for n inputs/outputs $\log(n)$ columns with $\frac{n}{2}$ rows of switches.

For each one of the $\frac{n}{2}\log(n)$ switches, we need $q + \log(n)$ copies to route q message bits together with the $\log(n)$ bits of their destination address. However, in the final column, we can remove the most significant address bit since it will no longer be required in the next stages of the network. For this reason, the depth $D_{nodeSW}(n)$ and the size $S_{nodeSW}(n)$ of the switches in a binary sorter are as follows for $n > 1$:

- $D_{nodeSW}(n) = \log(n)$
- $S_{nodeSW}(n) = (q + \log(n))\frac{n}{2}\log(n) - \frac{n}{2}$

As the final result, we therefore obtain the following depth $D_{node}(n)$ and size $S_{node}(n)$ for our binary sorter for $n > 1$:

- $D_{node}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ 1 & : \text{for } n = 2 \\ \log(n)(\log(n) - 1) + 2 & : \text{for } n > 2 \end{cases}$
- $S_{node}(n) = (q + \log(n) + 3)\frac{n}{2}\log(n) + \log(n) - \frac{9}{2}n + 4$

Hence, $D_{node}(n)$ is in $O(\log(n)^2)$ and the asymptotic complexity of $S_{node}(n)$ is in $O((q + \log(n))n\log(n))$ which means $O(n\log(n)^2)$ for constant q. While the size is exact, the depth computed above is just an upper bound of the real depth of the circuit (since the critical paths of the sequential parts of the circuit are not connected).

Complexity Analysis of the Network:
By the recursive construction of the radix-based sorting network as shown in Chapter 2, we directly derive the following formulas for the depths $D_{netCF}(n)$ and $D_{netSW}(n)$ and sizes $S_{netCF}(n)$ and $S_{netSW}(n)$ of the configuration logic and the switches, respectively:

- $D_{netCF}(n) = \begin{cases} D_{nodeCF}(n) + D_{netCF}\left(\frac{n}{2}\right) & : \text{for } n > 2 \\ 0 & : \text{for } n = 2 \end{cases}$
- $D_{netSW}(n) = \begin{cases} D_{nodeSW}(n) + D_{netSW}\left(\frac{n}{2}\right) & : \text{for } n > 2 \\ 1 & : \text{for } n = 2 \end{cases}$

- $S_{netCF}(n) = \begin{cases} S_{nodeCF}(n) + 2 \cdot S_{netCF}\left(\frac{n}{2}\right) & : \text{for } n > 2 \\ 0 & : \text{for } n = 2 \end{cases}$
- $S_{netSW}(n) = \begin{cases} S_{nodeSW}(n) + 2 \cdot S_{netSW}\left(\frac{n}{2}\right) & : \text{for } n > 2 \\ q & : \text{for } n = 2 \end{cases}$

and obtain the following solutions:

- $D_{netCF}(n) = \frac{1}{6}\left(2\log(n)^3 - 3\log(n)^2 + 7\log(n) - 6\right)$
- $D_{netSW}(n) = \frac{1}{2}\log(n)\left(\log(n) + 1\right)$
- $S_{netCF}(n) = \left(\frac{3}{4}n\log(n) - \frac{13}{4}n - 1\right)\log(n) + 6(n-1)$
- $S_{netSW}(q,n) = \frac{n}{12}\log(n)\left(2\log(n)^2 + (3q+3)\log(n) + 3q - 5\right)$

Hence, we conclude for the depth $D_{net}(n) = D_{netCF}(n) + D_{netSW}(n)$ and size $S_{net}(n) = S_{netCF}(n) + S_{netSW}(n)$ of entire interconnection network the following asymptotic complexities:

- $D_{net}(n) \in O(\log(n)^3)$
- $S_{net}(n) \in O((q + \log(n))n(\log(n))^2)$,
  i.e., $S_{net}(n) \in O(n(\log(n))^3)$ for constant q

Ranking-based Configuration

This circuit is based on the computation of ranks as suggested by [KoOr90] for the generalized cube network. Our contribution in this thesis is to show that a ranking-based approach can also configure the network of Figure 3.1 by using a parallel prefix computation similar to the one of the parities we used in Section 3.3.1 (using adders instead of XOR gates). To this end, we first compute the following ranks with the most significant target address bits $\mathsf{msb}(x_i)$:

$$r_i := \left(\sum_{j=0}^{i} \neg\mathsf{msb}(x_j)\right) - 1 \text{ for } i = 0, \ldots, n-1,$$

i.e., the number of inputs x_j in the prefix $x_0, \ldots, x_i$ that have a most significant bit $\mathsf{msb}(x_i) = 0$ minus 1. The ranks can now be used as local addresses of the network of Figure 3.1 in that an input x_i with $\mathsf{msb}(x_i) = 0$ must be routed to output y_{r_i}. We do not care about inputs x_i with $\mathsf{msb}(x_i) = 1$, since routing the inputs x_i with $\mathsf{msb}(x_i) = 0$ to the correct places will implicitly also route the others in the right part of the output sequence (maybe in a permuted ordering). This will make sure that the Split module sorts its inputs x_i by $\mathsf{msb}(x_i)$.

In general, permutation networks like the considered RB-FS-RV network are blocking, i.e., cannot implement all permutations of input addresses to

Table 3.2.: *Ranking-based configuration p_i of switch i in the network shown in Figure 3.1 based on the most significant bits $\mathsf{msb}(x_{2i})$,$\mathsf{msb}(x_{2i+1})$ of the target addresses and the ranks ranks r_{2i}, r_{2i+1} of the inputs x_{2i}, x_{2i+1}, respectively. We also determine the local addresses/ranks rL_i and rU_i to be used in the lower and upper subnetworks for further routing.*

$\mathsf{msb}(x_{2i})$	$\mathsf{msb}(x_{2i+1})$	r_{2i}	r_{2i+1}	p_i	rL_i	rU_i
0	0	$2k$	$2k+1$	$*$	k	k
0	0	$2k+1$	$2k+2$	1	$k+1$	k
0	1	$2k$	–	0	k	–
0	1	$2k+1$	–	1	–	k
1	0	–	$2k$	1	k	–
1	0	–	$2k+1$	0	–	k
1	1	–	–	$*$	–	–

target addresses. However, the RB-FS-RV network is able to route the 0s to monotonically increasing target addresses as proved in the following. *We will also see that the ranking-based approach will compute exactly the same configuration as the approaches using parity bits.* To this end, consider Table 3.2 where we determine the configuration p_i of switch i based on the most significant bits $\mathsf{msb}(x_{2i})$ and $\mathsf{msb}(x_{2i+1})$ of the target addresses of inputs x_{2i} and x_{2i+1}, respectively, and their ranks r_{2i} and r_{2i+1}, respectively.

The configuration p_i of switch i is obtained according to Table 3.2 as follows: First note that the lower subnetwork in Figure 3.1 is connected with the even output addresses, and that the upper subnetwork is connected with the odd output addresses. Second, recall that rank $r_i = k$ means that there are $k+1$ 0s in the prefix $x_0,\ldots,x_k$. Thus, we have the following cases (note that we do not care about routing inputs x_i with $\mathsf{msb}(x_i) = 1$ since these are implicitly routed to the remaining target addresses):

- If $\mathsf{msb}(x_{2i}) = \mathsf{msb}(x_{2i+1}) = 0$ and $r_{2i} = 2k$ holds, then $r_{2i+1} = r_{2i} + 1 = 2k+1$ holds, and the prefix $x_0,\ldots,x_{2i-1}$ contains $(r_{2i}-1)+1 = 2k$ 0s which are equally distributed, i.e., k of them have been sent to the lower subnetwork and k others to the upper subnetwork that occupy the local addresses $0,\ldots,k-1$ there. The configuration p_i does not matter here, we will send one input to the lower and the other to the upper subnetwork with local address/rank $k = \lfloor \frac{r_{2i}}{2} \rfloor = \lfloor \frac{r_{2i+1}}{2} \rfloor$.

- If $\mathsf{msb}(x_{2i}) = \mathsf{msb}(x_{2i+1}) = 0$ and $r_{2i} = 2k+1$ holds, then $r_{2i+1} = r_{2i} + 1 = 2k+2$ holds, and the prefix $x_0,\ldots,x_{2i-1}$ contains $(r_{2i}-1)+1 = 2k+1$ 0s which are equally distributed, i.e., $k+1$ of them have been sent to the lower subnetwork and k others to the upper subnetwork that occupy the local addresses $0,\ldots,k$ and $0,\ldots,k-1$ there. The configuration p_i does matter here in the sense that we will send 0s to each subnetwork, but we have to define $p_i := 1$ so that x_{2i} will be sent to the upper subnetwork with local address/rank $k = \lfloor \frac{r_{2i}}{2} \rfloor = \frac{2k+1}{2}$ while x_{2i+1} will be sent to the lower subnetwork with local address/rank $k+1 = \lfloor \frac{r_{2i+1}}{2} \rfloor = \frac{2k+2}{2}$.

- If $\mathsf{msb}(x_{2i}) = 0$, $\mathsf{msb}(x_{2i+1}) = 1$, and $r_{2i} = 2k$ holds, then the prefix $x_0,\ldots,x_{2i-1}$ contains $(r_{2i}-1)+1 = 2k$ 0s which are equally distributed, i.e., k of them have been sent to the lower subnetwork and k others to the upper subnetwork that occupy the local addresses $0,\ldots,k-1$ there. We define $p_i := 0$ so that x_{2i} will be

sent to the lower subnetwork with local address/rank $k = \lfloor \frac{r_{2i}}{2} \rfloor = \frac{2k}{2}$ while x_{2i+1} will be sent to the upper subnetwork (and we don't care about its address).

- If $\mathsf{msb}(x_{2i}) = 0$, $\mathsf{msb}(x_{2i+1}) = 1$, and $r_{2i} = 2k+1$ holds, then the prefix $x_0,\ldots,x_{2i-1}$ contains $(r_{2i}-1)+1 = 2k+1$ 0s which are equally distributed, i.e., $k+1$ of them have been sent to the lower subnetwork and k others to the upper subnetwork that occupy the local addresses $0,\ldots,k$ and $0,\ldots,k-1$ there. We define $p_i := 1$ so that x_{2i} will be sent to the upper subnetwork with local address/rank $k = \lfloor \frac{r_{2i}}{2} \rfloor = \frac{2k+1}{2}$ while x_{2i+1} will be sent to the lower subnetwork (and we don't care about its address).

- If $\mathsf{msb}(x_{2i}) = 1$, $\mathsf{msb}(x_{2i+1}) = 0$, and $r_{2i+1} = 2k$ holds, then the prefix $x_0,\ldots,x_{2i-1}$ contains $(r_{2i+1}-1)+1 = 2k$ 0s which are equally distributed, i.e., k of them have been sent to the lower subnetwork and k others to the upper subnetwork that occupy the local addresses $0,\ldots,k-1$ there. We define $p_i := 1$ so that x_{2i+1} will be sent to the lower subnetwork with local address/rank $k = \lfloor \frac{r_{2i+1}}{2} \rfloor = \frac{2k}{2}$ while x_{2i} will be sent to the upper subnetwork (and we don't care about its address).

- If $\mathsf{msb}(x_{2i}) = 1$, $\mathsf{msb}(x_{2i+1}) = 0$, and $r_{2i+1} = 2k+1$ holds, then the prefix $x_0,\ldots,x_{2i-1}$ contains $(r_{2i+1}-1)+1 = 2k+1$ 0s which are equally distributed, i.e., $k+1$ of them have been sent to the lower subnetwork and k others to the upper subnetwork that occupy the local addresses $0,\ldots,k$ and $0,\ldots,k-1$ there. We define $p_i := 0$ so that x_{2i+1} will be sent to the upper subnetwork with local address/rank $k = \lfloor \frac{r_{2i+1}}{2} \rfloor = \frac{2k+1}{2}$ while x_{2i} will be sent to the lower subnetwork (and we don't care about its address).

- The only not yet considered cases are those where $\mathsf{msb}(x_{2i}) = \mathsf{msb}(x_{2i+1}) = 1$ hold with arbitrary ranks. Since we do not care about routing these inputs, the configuration p_i can be chosen arbitrarily in these cases, and we also do not have to determine local addresses/ranks for these inputs.

Note that the above discussed cases which are also listed in Table 3.2 are complete, since in the first two cases, r_{2i+1} is determined by the values of $\mathsf{msb}(x_{2i}), \mathsf{msb}(x_{2i+1}), r_{2i}$. For this reason, we now derive the following Karnaugh-Veitch diagram from Table 3.2 using the least significant bits $\mathsf{lsb}(r_{2i})$, $\mathsf{lsb}(r_{2i+1})$ of r_{2i},r_{2i+1}, respectively:

	$\mathsf{msb}(x_{2i+1})\,\mathsf{lsb}(r_{2i+1})$			
$\mathsf{msb}(x_{2i})\,\mathsf{lsb}(r_{2i})$	00	01	11	10
00	*	*	0	0
01	1	*	1	1
11	1	0	*	*
10	1	0	*	*

We therefore can define p_i by the following minimal disjunctive normal forms:

- $p_i := \mathsf{msb}(x_{2i}) \wedge \neg\mathsf{lsb}(r_{2i+1}) \vee \neg\mathsf{msb}(x_{2i}) \wedge \mathsf{lsb}(r_{2i})$
- $p_i := \mathsf{msb}(x_{2i+1}) \wedge \mathsf{lsb}(r_{2i}) \vee \neg\mathsf{msb}(x_{2i+1}) \wedge \neg\mathsf{lsb}(r_{2i+1})$
- $p_i := \neg\mathsf{msb}(x_{2i}) \wedge \mathsf{lsb}(r_{2i}) \vee \neg\mathsf{msb}(x_{2i+1}) \wedge \neg\mathsf{lsb}(r_{2i+1})$

Using multiplexers, we can also equivalently define

- $p_i := \mathsf{if}\ \mathsf{msb}(x_{2i})\ \mathsf{then}\ \neg\mathsf{lsb}(r_{2i+1})\ \mathsf{else}\ \mathsf{lsb}(r_{2i})$
- $p_i := \mathsf{if}\ \mathsf{msb}(x_{2i+1})\ \mathsf{then}\ \mathsf{lsb}(r_{2i})\ \mathsf{else}\ \neg\mathsf{lsb}(r_{2i+1})$

Having determined the configurations p_i of switch i in the first column of the network shown in Figure 3.1, we can recursively determine the configurations of the other columns in the subnetworks since we have also determined the local addresses/ranks rL_i and rU_i to be used in the subnetworks as shown in Table 3.2: As can be seen, the local address/rank rL_i and rU_i is simply obtained by removing the least significant bit from the rank of the value that is routed to that place.

Using ranks as local target addresses for the Split modules determines the same configurations that we obtained by computing parity bits: Note that all inputs x_i with $\mathsf{msb}(x_i) = 0$ were routed to the lower subnetwork if their rank was even, and otherwise to the upper subnetwork. Hence, also the ranking-based approach equally distributes the inputs x_i with $\mathsf{msb}(x_i) = 0$ and $\mathsf{msb}(x_i) = 1$, respectively, to the lower and upper subnetworks, given preference to the lower and upper subnetworks for additional inputs x_i with $\mathsf{msb}(x_i) = 0$ and $\mathsf{msb}(x_i) = 1$, respectively.

Complexity Analysis of the Binary Sorter:
For computing the configuration of the switches in Figure 3.1, we have to compute the following ranks with the most significant target address bits $\mathsf{msb}(x_i)$:

$$r_i := \left(\sum_{j=0}^{i} \neg\mathsf{msb}(x_j)\right) - 1 \text{ for } i = 0, \ldots, n-1,$$

i.e., the number of inputs x_j in the prefix $x_0, \ldots, x_i$ that have a most significant bit $\mathsf{msb}(x_i) = 0$ minus 1. These ranks can be computed as a parallel prefix computation similar to the one of the parities we used in Section 3.3.1 (using adders instead of XOR gates).

Since the ranking circuit is based on a parallel prefix computation, we can compute the depth $D_{rank}(n)$ and size $S_{rank}(n)$ similarly to Section 3.3.1 (using adders instead of XOR gates)

- $D_{rank}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ 1 & : \text{for } n = 2 \\ 2\log(n) - 2 & : \text{for } n > 2 \end{cases}$
- $S_{rank}(n) = 2n - \log(n) - 2$ for all n that are powers of 2

The size of the adders used ranges from one bit adders on the lowest level up to $\log(n)$ bit address at the highest level. Hence, as an upper bound, no more than $\log(n)(2n - \log(n) - 2)$ gates were used.

Having computed the ranks, we can compute the configurations p_i of all the switches in the first column of the binary sorter for $i \in \{0, \ldots, \frac{n}{2} - 1\}$ using $\frac{n}{2}$ multiplexer (see section 3.3.2). Hence, the circuit depth $D_{colCF}(n)$ and size $S_{colCF}(n)$ for computing the all p_i for $i \in \{0, \ldots, \frac{n}{2} - 1\}$ of one column are as follows:

- $D_{colCF}(n) = \begin{cases} 0 & : \text{for } n \leq 2 \\ D_{rank}(n) + 1 & : \text{for } n > 2 \end{cases}$

- $S_{colCF}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ S_{rank}(n) + \frac{n}{2} & : \text{for } n > 1 \end{cases}$

which means the following for all n that are powers of 2:

- $D_{colCF}(n) = \begin{cases} 0 & : \text{for } n \leq 2 \\ 2 & : \text{for } n = 4 \\ 2\log(n) - 1 & : \text{for } n > 4 \end{cases}$

- $S_{colCF}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ \frac{n}{2} - 2\log(n)(n-1) - \log(n)^2 & : \text{for } n > 1 \end{cases}$

Having determined the configurations of the first column of the network, we can recursively determine the configurations of the other columns in the subnetworks using the rank as the local address. As can be seen, the local address/rank for the subnetworks is simply obtained by removing the least significant bit from the rank of the value that is routed to that place. Therefore, to compute the depth $D_{nodeCF}(n)$ and size $S_{nodeCF}(n)$ of the entire configuration logic for the binary sorter shown in Figure 3.1, we set up the following formulas:

- $D_{nodeCF}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ D_{colCF}(n) + (\log n - 1) & : \text{for } n > 1 \end{cases}$

- $S_{nodeCF}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ S_{colCF}(n) + \frac{n}{2}.(\log n - 1) & : \text{for } n > 1 \end{cases}$

and which have the following solutions:

- $D_{nodeCF}(n) = \begin{cases} 0 & : \text{for } n \leq 2 \\ 3\log(n) - 2 & : \text{for } n > 2 \end{cases}$

- $S_{nodeCF}(n) = \left(\frac{5}{2}n - 2 - \log(n)\right)\log(n) \text{ for } n \geq 1$

In addition to the above size and depth for the configuration logic, we also have to add the switches needed in the binary sorter. It is easily seen that the recursive definition according to Figure 3.1 will generate for n inputs/outputs $\log(n)$ columns with $\frac{n}{2}$ rows of switches. For each one of the $\frac{n}{2}\log(n)$ switches, we need $q + \log(n)$ copies to route q message bits together with the $\log(n)$ bits of their destination address. In addition to $q + \log(n)$ bits, each switch in the i^{th} column carries $\log(n) - i - 1$ bits of rank information. In the final column, we can remove the most significant address bit since it will no longer be required in the next stages of the network. For this reason, the depth $D_{nodeSW}(n)$ and the size $S_{nodeSW}(n)$ of the switches in a binary sorter are as follows for $n > 1$:

- $D_{nodeSW}(n) = \log(n)$

- $S_{nodeSW}(n) = (q + \log(n))\frac{n}{2}\log(n) - \frac{n}{2} + \sum_{i=0}^{\log(n)-1}(\log(n) - i - 1))$

As the final result, we therefore obtain the following depth $D_{node}(n)$ and size $S_{node}(n)$ for the binary sorter for $n > 1$:

- $D_{node}(n) = \begin{cases} 0 & : \text{for } n = 1 \\ 1 & : \text{for } n = 2 \\ 3\log(n) - 2 + \log(n) & : \text{for } n > 2 \end{cases}$
- $S_{node}(n) = (4q + 5\log(n) - 1)\frac{n}{4}\log(n) - \frac{n}{2}$

Hence, $D_{node}(n)$ is in $O(\log(n))$ and the asymptotic complexity of $S_{node}(n)$ is in $O((4q + 5\log(n) - 1)\frac{n}{4}\log(n))$ which means $O(n\log(n)^2)$ for constant q.

It can be easily seen that size and depth of ranking based circuit to compute the configuration logic of presented split module is asymptotically better than the parallel parity computation circuit.

Complexity Analysis of the Network:
By the recursive construction of the radix-based sorting network as shown in Chapter 2, we directly derive the following formulas for the depths $D_{netCF}(n)$ and $D_{netSW}(n)$ and sizes $S_{netCF}(n)$ and $S_{netSW}(n)$ of the configuration logic and the switches, respectively:

- $D_{netCF}(n) = \begin{cases} D_{nodeCF}(n) + D_{netCF}\left(\frac{n}{2}\right) & : \text{for } n > 2 \\ 0 & : \text{for } n = 2 \end{cases}$
- $D_{netSW}(n) = \begin{cases} D_{nodeSW}(n) + D_{netSW}\left(\frac{n}{2}\right) & : \text{for } n > 2 \\ 1 & : \text{for } n = 2 \end{cases}$
- $S_{netCF}(n) = \begin{cases} S_{nodeCF}(n) + 2 \cdot S_{netCF}\left(\frac{n}{2}\right) & : \text{for } n > 2 \\ 0 & : \text{for } n = 2 \end{cases}$
- $S_{netSW}(n) = \begin{cases} S_{nodeSW}(n) + 2 \cdot S_{netSW}\left(\frac{n}{2}\right) & : \text{for } n > 2 \\ q & : \text{for } n = 2 \end{cases}$

and obtain the following solutions:

- $D_{netCF}(n) = \frac{3}{2}\left(\log(n)^2 + \log(n) - \frac{4}{3}\right)$
- $D_{netSW}(n) = \frac{1}{2}\log(n)(\log(n) + 1)$
- $S_{netCF}(n) = \frac{5}{2}\log(n)^2\left(n + \frac{2}{5}\right) + \frac{5}{2}\log(n)\left(n + \frac{12}{5}\right) - 10(n-1)$
- $S_{netSW}(q,n) = \frac{5n}{12}\log(n)^3 + 2nq\log(n)^2 + \frac{n}{2}\log(n)(4q+1)$

Hence, we conclude for the depth $D_{net}(n) = D_{netCF}(n) + D_{netSW}(n)$ and size $S_{net}(n) = S_{netCF}(n) + S_{netSW}(n)$ of entire interconnection network the following asymptotic complexities:

- $D_{net}(n) \in O(\log(n)^2)$
- $S_{net}(n) \in O(n((\log(n))^3))$

Thus, the final size of the entire RBS network is asymptotically the same for these two circuits, namely $O(n\log(n)^3)$, the depth of the RBS network with the ranking-based approach is only $O(\log(n)^2)$ while it is $O(\log(n)^3)$ for the parallel parity computation.

3.3. Sorters with Half Cleaner based RBS Network

Fortunately, Narasimha observed in [Nara88] (Section V) that one can construct *Split* modules as shown in Figure 3.4 using Batcher's sorting networks with $\frac{n}{2}$ inputs/outputs and a half cleaner circuit. Half cleaners were introduced by Batcher in [Batc68] for the construction of his bitonic sorting networks. Narasimha's construction has also been implicitly used in [KoOr90], but no correctness proofs were reported so far. To that end, we found a simple and general lemma about half cleaner modules (can be derived from Batcher's original lemma) that can also be used with any binary and ternary sorters to implement binary and ternary[4] Split modules.

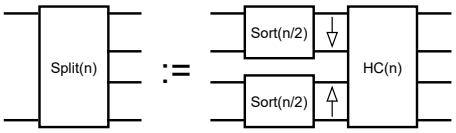


Figure 3.4.: *Implementing Split modules by two sorting networks and a half cleaner module.*

3.3.1. The Half Cleaner Lemma

We first define half cleaner and then prove the half cleaner lemma that is used to implement binary and ternary (see next chapter) Split modules.

Sorting networks are usually constructed by compare-and-swap switches which have two inputs x_0 and x_1 that are routed to the two outputs y_0 and y_1 such that $y_0 := \min\{x_0, x_1\}$ and $y_1 := \max\{x_0, x_1\}$ holds. The compare-and-swap switch, shown in Figure 3.5, uses an arrow to denote the minimum output y_0.

Definition 3.6 ⟨ Half Cleaner ⟩
A half cleaner with $2n$ inputs $x_0, \ldots, x_{2n-1}$ of some totally ordered set and outputs $y_0, \ldots, y_{2n-1}$ consists of n compare-and-swap switches such that

[4] For partial permutation (see the next chapter), we have to replace the binary sorters by ternary sorters, since we then find target addresses with most significant bits $0, \perp, 1$ where we use $\perp$ as most significant bit of those inputs that have no associated target address.

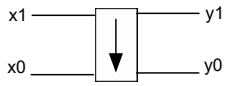


Figure 3.5.: *Compare-and-swap switch (the arrow points towards the minimum output).*

switch i has inputs x_i and x_{i+n} and outputs y_i and y_{i+n} that are computed as $y_i := \min\{x_i, x_{n+i}\}$ and $y_{n+i} := \max\{x_i, x_{n+i}\}$ for $i = 0, \ldots, n-1$.

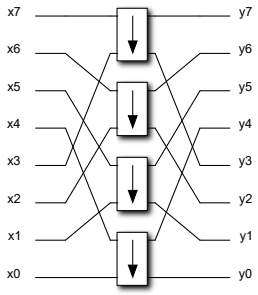


Figure 3.6.: *A half cleaner module with 8 inputs and 8 outputs consisting of four 2×2 compare-and-swap switches (the arrows of compare-and-swap switches point towards the minimum output).*

Half cleaner modules were introduced in [Batc68] as building blocks for Batcher's bitonic sorting network. To prove its correctness, Batcher proved the following Lemma in [Batc68]:

Lemma 3.1 (Half Cleaner Lemma (I)) *Given a bitonic[5] sequence $x_0, \ldots, x_{2n-1}$ with elements x_i of some totally ordered set as input to a half cleaner, the following holds for its outputs $y_0, \ldots, y_{2n-1}$:*

- *$y_i \leq y_{n+j}$ for all $i, j \in \{0, \ldots, n-1\}$ and*
- *both halves $y_0, \ldots, y_{n-1}$ and $y_n, \ldots, y_{2n-1}$ are bitonic sequences.*

Batcher used the above lemma for the recursive construction of the sorting network. We prove a simpler lemma about the half cleaner module for ternary inputs, i.e., $0, \perp, 1$ where we use $\perp$ as inputs that have no associated target address. This half cleaner lemma will be useful for implementing both the binary and ternary[6] Split modules. Our lemma is the following one:

[6]For partial permutation (see the next chapter), we have to replace the binary sorters by

Lemma 3.2 (Half Cleaner Lemma (II)) *Given sorted lists* $a_0 \preceq \ldots \preceq a_{n-1}$ *and* $b_0 \succeq \ldots \succeq b_{n-1}$ *with elements* $a_i, b_i \in \{0, \perp, 1\}$ *with the total order* $0 \preceq \perp \preceq 1$ *such that the following holds (i.e., both the numbers of 0s and 1s contained are* $\leq n$*):*

- $|\{i \in \{0, \ldots, n-1\} \mid a_i = 0\}| + |\{i \in \{0, \ldots, n-1\} \mid b_i = 0\}| \leq n$
- $|\{i \in \{0, \ldots, n-1\} \mid a_i = 1\}| + |\{i \in \{0, \ldots, n-1\} \mid b_i = 1\}| \leq n$

If the input sequence $a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}$ *is given as input to a half cleaner, we obviously obtain the following outputs* $y_0, \ldots, y_{2n-1}$ *for* $i = 0, \ldots, n-1$*:*

- $y_i := \min\{a_i, b_i\}$ *and*
- $y_{n+i} := \max\{a_i, b_i\}$

It then follows that we have $y_i \in \{0, \perp\}$ *and* $y_{n+i} \in \{\perp, 1\}$ *for* $i = 0, \ldots, n-1$*, so that the left half* $y_0, \ldots, y_{n-1}$ *contains all values* 0 *while the right half* $y_n, \ldots, y_{2n-1}$ *contains all values* 1 *of the input lists. Exchanging input sequences* $a_0 \preceq \ldots \preceq a_{n-1}$ *and* $b_0 \succeq \ldots \succeq b_{n-1}$ *with each other yields the same results.*

Proof: We first prove that the two values a_i and b_i that arrive at a switch of the half cleaner can be neither both 0 nor can they be both 1:

- Assuming that both a_i and b_i would be 0, it would follow that at least all $a_0, \ldots, a_i$ and also all $b_i, \ldots, b_{n-1}$ would have to be 0 since the input lists a and b are sorted. However, these are $(i+1)+(n-i) = n+1$ values, which is in contradiction to the assumption that at most n values are 0.
- Assuming that both a_i and b_i would be 1, it would follow that at least all $a_i, \ldots, a_{n-1}$ and also all $b_0, \ldots, b_i$ would have to be 1 since the input lists a and b are sorted. However, these are $(n-i)+(i+1) = n+1$ values, which is in contradiction to the assumption that at most n values are 1.

Hence, it is impossible that two input values a_i, b_i of switch i in the half cleaner would be both 0 or both 1 (but both could be $\perp$). Therefore, $y_i := \min\{a_i, b_i\} \in \{0, \perp\}$ and $y_{n+i} := \max\{a_i, b_i\} \in \{\perp, 1\}$ for $i = 0, \ldots, n-1$ as can be seen by the table in Figure 3.7. ■

Now, we will show how the above lemma can be used to implement a *Split* module for the construction of an *RBS* network.

ternary sorters, since we then find target addresses with most significant bits $0, \perp, 1$ where we use $\perp$ as most significant bit of those inputs that have no associated target address.

a_i	b_i	$y_i := \min\{a_i, b_i\}$	$y_{n+i} := \max\{a_i, b_i\}$
~~$\emptyset$~~	~~$\emptyset$~~	~~$\emptyset$~~	~~$\emptyset$~~
0	$\perp$	0	$\perp$
0	1	0	1
$\perp$	0	0	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$
$\perp$	1	$\perp$	1
1	0	0	1
1	$\perp$	$\perp$	1
~~1~~	~~1~~	~~1~~	~~1~~

Figure 3.7.: *Possible/impossible inputs and outputs of compare-and-swap switches of the half cleaner under the assumptions given in Lemma 3.2: The first and the last input/output rows cannot occur, and therefore $y_i \neq 1$ and $y_{i+n} \neq 0$ holds for $i = 0, \ldots, n-1$.*

Theorem 3.1 (Half Cleaner Optimization of Split Modules)
Given an input sequence $x_0, \ldots, x_{2n-1}$ where $x_i \in \{0, \perp, 1\}$ and where at most n of the inputs x_i are 0 and also at most n of the inputs x_i are 1, then the outputs $y_0, \ldots, y_{2n-1}$ of the Split module shown in Figure 3.4 will satisfy the following:

- *$y_i \in \{0, \perp\}$ for $i = 0, \ldots, n-1$*
- *$y_{n+i} \in \{\perp, 1\}$ for $i = 0, \ldots, n-1$*

Hence, all $x_i = 0$ are routed to the lower half $y_0, \ldots, y_{n-1}$ and all $x_i = 1$ are routed to the upper half $y_n, \ldots, y_{2n-1}$.

Proof: Given a ternary input sequence $x_0, \ldots, x_{2n-1}$, we can sort its lower half $x_0, \ldots, x_{n-1}$ and its upper half $x_n, \ldots, x_{2n-1}$ separately using the sorting networks shown in Figure 3.4. The outputs of the lower and upper sorting networks are the sequences $b_0 \succeq \ldots \succeq b_{n-1}$ and $a_0 \preceq \ldots \preceq a_{n-1}$ mentioned in Lemma 3.2, respectively. Thus, the above theorem follows now directly from Lemma 3.2. ■

According to the above theorem, the output sequence $y_0, \ldots, y_{2n-1}$ is partitioned such that the 0s are in the lower half, the 1s are in the upper half, and inputs $\perp$ may occur in both halves. Hence, the half cleaner and the two sorting networks implement a *Split* module as required for the recursive construction of a RBS network even in case of partial permutations.

Figure 3.8 shows a general RBS network for 8 inputs constructed by *Split* modules according to Figure 3.4. Since the size and the depth of the these half cleaner circuits are $O(n)$ and $O(1)$, respectively, the size and depth of these *Split* modules are mainly dominated by the used sorting networks.

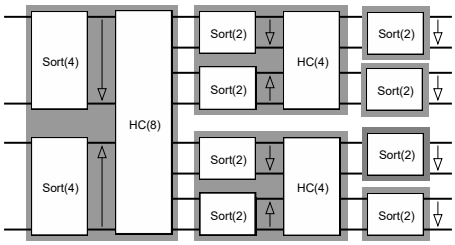


Figure 3.8.: *A RBS network for 8 inputs constructed by Split modules according to Figure 3.4.*

Complexity Analysis of the split module and the Network:
The complexity of this network mainly depends up on the used binary sorter to implement the *Split* modules. Clearly, we can compute the depth $D_{hc}(n) := D(\frac{n}{2})+1$ and the size $S_{hc}(n) := 2 \cdot S(\frac{n}{2})+c \cdot n$ of these *Split* modules for a binary sorter of depth $D(n)$ and size $S(n)$. For example, for a depth $D(n) := a \cdot \log(n)^b$ this becomes $D_{hc}(n) = a \cdot (\log(n)-1)^b+1 \in \Theta(a \cdot \log(n)^b)$ thus becomes negligible for large n, while for a depth $D(n) := a \cdot n$ (like in [Nara94]), this becomes $D_{hc}(n) = \frac{a}{2}n+1$ and roughly halvens the depth. For a size $S(n) := a \cdot n \cdot \log(n)^b$ this becomes $S_{hc}(n) = a \cdot n \cdot (\log(n)-1)^b + c \cdot n \in \Theta(a \cdot n \cdot \log(n)^b)$, thus also negligible for large n. Hence, this half cleaner construction does not improve the asymptotic complexity of neither the considered binary sorter nor the network implemented using this construction.

3.4. Summary

In this chapter, we present new non-blocking unicast interconnection networks based on radix sorting scheme. As any interconnection network based on radix sorting, also these networks mainly depend on the construction of a suitable *Split* module.

The core idea of the proposed *Split* modules is the use of the permutation network of Narasimha [Nara94]. Therefore, first two *Split* modules are based on a reverse-banyan flip-shuffle permutation network with an outgoing permutation that reverses the addresses. Narasimha showed that using the parities $p_i := x_0 \oplus \ldots \oplus x_{2i}$ to configure the switches will make the *Split* module a binary sorter. Therefore, we present two circuits to compute the switch configurations $p_i := x_0 \oplus \ldots \oplus x_{2i}$ as required to implement a *Split* module. The first circuit is based on a work- efficient parallel prefix computation and the second is based on a ranking-based approach.

Since ranks can be computed by a parallel prefix sum using $O(n)$ gates and $O(\log(n))$ depth, the size and depth of the ranking-based circuit to compute the configuration logic is asymptotically better than the parallel prefix computation. However, while the ranking-based approach improves the size and depth of the configuration logic, it has the disadvantage that the computed ranks have to be forwarded to the subnetworks which requires additional switches. Thus, while saving gates for computing the configuration, we have

to add gates for forwarding the centrally computed configuration (the ranks) through the network.

While the final size of the entire RBS network is asymptotically the same for these two circuits, namely $O(n \log(n)^3)$, the depth of the RBS network with the ranking-based approach is only $O(\log(n)^2)$ while it is $O(\log(n)^3)$ for the parallel parity computation.

The third network uses two sorters with a half cleaner to implement the *Split* module. Since the size and the depth of the half cleaner circuits to implement this third RBS network is $O(n)$ and $O(1)$ respectively, the size and the depth of this RBS network are mainly dominated by the used sorting networks.

Chapter 4

New RBS networks for Partial Permutations

Contents

4.1. Routing Partial Permutations by Sorting Networks 47
4.2. Routing Partial Permutations 49
4.2.1. Front-end Valid Sorter for Prefix Defined Networks 49
4.2.2. Constructing Split Modules as Ternary Sorters . . . 52
4.2.3. Constructing Split Modules by Ternary Concentrators 53
4.2.4. Constructing Split Modules by Ternary Sorters and Half Cleaners 54
4.3. Summary 54

Nonblocking unicast interconnection networks allow every input component $x_0,\ldots,x_{n-1}$ to be connected with any output component $y_0,\ldots,y_{n-1}$ provided that none of the outputs y_j is the target of more than one input x_i. Hence, such networks can implement *all* $n!$ *permutations* of the addresses $\{0,\ldots,n-1\}$ as routes through a switching network. In practice, however, not all input components have to be always connected to an output component. For this reason, even *all* $\sum_{i=0}^{n} i!\binom{n}{i}^2$ *partial permutations* have to be implemented by non-blocking unicast interconnection networks.

As already mentioned, sorting networks [Batc68] are an attractive alternative for the design of non-blocking interconnection networks [GaPa83]. However, the implementation of partial permutations by sorting networks is not straightforward and depends on the used sorting algorithms as we will outline in the next section.

4.1. Routing Partial Permutations by Sorting Networks

Independent on the choice of a particular sorting algorithm, sorting networks at first only implement *total permutations* in that they can sort the n inputs

by their target addresses which are numbers $0, \ldots, n-1$. If some inputs do not need a connection to an output, their target addresses are invalid, denoted as $\perp$ in the following. Note that there is no ordering of $\{\perp, 0, \ldots, n-1\}$ that would still solve the routing problem by a simple sorting approach, since many values $\perp$ may now occur and they may have to be routed to different places in the final output sequence.

For merge-based sorting networks, there is a well-known solution known as the Batcher-Banyan network [HuKn84; Nara88]. The main idea is thereby to first treat $\perp$ as a number larger than all target addresses so that after using a normal sorting network this way, one obtains a preliminary output sequence $y_0, \ldots, y_{k-1}, y_k, \ldots, y_{n-1}$ where the k valid inputs were sorted as the prefix $y_0, \ldots, y_{k-1}$ while the invalid ones are placed in the suffix $y_k, \ldots, y_{n-1}$. A final Banyan permutation network can then be used to move the valid inputs $y_0, \ldots, y_{k-1}$ to the right places. To that end, one can simply use a bit-controlled network like the Ω-network [Lawr75] where invalid target addresses $\perp$ are ignored, so that the valid ones are routed to their final destination. It can be shown [Nara88] that the Ω-network [Lawr75] while being blocking in general will never block in this setting.

The same approach does however not work for the radix-based networks: If we treat $\perp$ as a number larger than all target addresses, it may happen that valid inputs with a most significant bit 1 will be erroneously routed by Split modules to the lower sub-network, where they are mixed up with other valid inputs having a most significant bit 0. Hence, the resulting preliminary output sequence will not consist of a sorted prefix of valid inputs as in the case of merge-based networks.

Hence, the Batcher-Banyan construction does not work for RBS networks. Recall that the task of Split modules was to route the inputs already in the right halves. Inputs with invalid target addresses can be routed to any half, but inputs with valid target address must be routed to the lower and upper sub-networks in case the most significant bit of the target address is 0 and 1, respectively.

Instead of using binary sorters as in case of total permutations, one could therefore use *ternary sorters* as Split modules using the ordering $0 \preceq \perp \preceq 1$. This way, the output sequence of a Split module will still route the inputs with valid target addresses to the right halves, while invalid inputs may be routed to any half (note that still at most $\frac{n}{2}$ inputs can have most significant bits 0/1). However, while many constructions for binary sorters have been proposed [JaOr93; LeOr95a; ChOr94; ChCh96; KoOr90; Nara94], none are known for the ternary case.

Narasimha addressed the problem to route partial permutations in his RBS network in [Nara94]. In Section III of [Nara94], he explains that his network can also work with partial permutations if an additional Split module is added on the left side of his RBS network. However, he neither gave a proof of this claim, nor did he discuss how that construction could be generalized to other networks.

Therefore, in this chapter, we give a formal proof of this statement and gen-

eralize it to an entire class of configuration circuits. Additionally, also present three general constructions that can transform binary sorters to ternary Split modules. This way, one can transform any RBS network that has been constructed for total permutations into a more powerful one that can work with partial permutations as well.

4.2. Routing Partial Permutations

For all networks with partial permutations, we assume that any input x_i is a bitvector in the format given as below: The leftmost q bits $x_{i,0} \dots x_{i,q-1}$ are the *message bits* that should be sent to an output, $x_{i,q}$ is the *valid bit* that indicates whether this input contains a message and shall be connected to some output, and the remaining bits $x_{i,q+1} \dots x_{i,q+\log(n)}$ are the *bits of the target address* where $x_{i,q+\log(n)}$ is the most significant bit.

$x_{i,0} \dots x_{i,q-1}$	$x_{i,q}$	$x_{i,q+1} \dots x_{i,q+\log(n)}$

For partial permutations, the valid bit $x_{i,q}$ and bit $x_{i,q+\log(n)}$ are now considered together as most significant bit of input x_i in that we define the following value $\mathsf{msb}(x_i) \in \{0, \perp, 1\}$ by $x_{i,q}$ and $x_{i,q+\log(n)}$ as follows:

$x_{i,q}$	$x_{i,q+\log(n)}$	$\mathsf{msb}(x_i)$
0	$*$	$\perp$
1	0	0
1	1	1

Therefore, we consider now inputs x_i of the Split modules that have $\mathsf{msb}(x_i) \in \{0, \perp, 1\}$ where $\perp$ means that the x_i has no valid target address, while 0 and 1 are the most significant bits of the otherwise valid target address. In every stage of the RBS networks, the Split modules will remove a target address bit $x_{i,q+\log(n)}$ in the RBS network so that finally only the message and the valid bit will arrive at the output.

4.2.1. Front-end Valid Sorter for Prefix Defined Networks

A major observation made by Narasimha [Nara94] was that his RBS network also works for partial permutations if an *additional Split module (called a front-end concentrator)* is added (see Figure 4.1 and 4.3 for the general construction and Figure 4.2 for $n = 8$ inputs). Such a additional Split module(also called a valid sorter) is also a RB-FS-RV network that is used as a special binary sorter: To this end, we *simply sort the inputs x_i according to their negated valid bits $x_{i,q}$*. This means that the invalid messages are sorted to a suffix of the sequence. As a result, the vector u_i that is generated as output from the front-end valid sorter is a prefix sequence defined as follows:

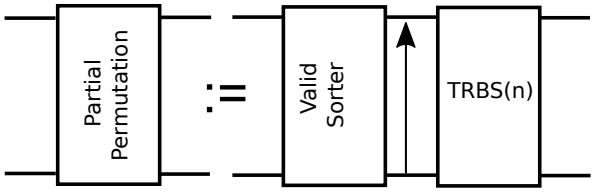


Figure 4.1.: *Converting Sequences to Prefix Sequences.*

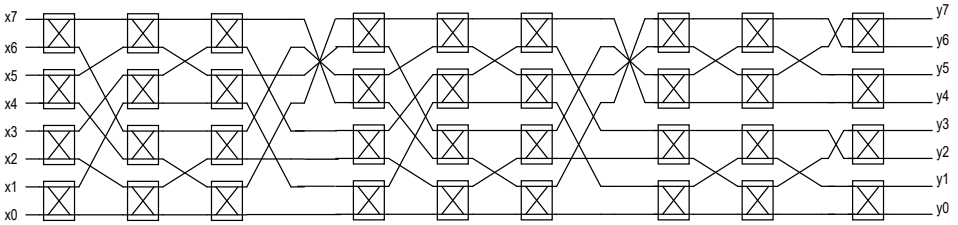


Figure 4.2.: *Entire Interconnection Network based on Radix-based Sorting.*

Definition 4.1 ⟨ Prefix Sequence ⟩
Sequence $x_0, \ldots, x_{n-1}$ is called a prefix sequence, if the sequence $\mathsf{msb}(x_0), \ldots, \mathsf{msb}(x_{n-1})$ of their most significant bits is of the form $b_0, \ldots, b_k, \perp, \ldots, \perp$ with $b_i \in \{0, 1\}$.

The task of the front-end valid sorter is therefore to partition the inputs into valid and invalid ones which can be easily done by sorting according to $\neg x_{i,q}$. We will prove next that the configurations we determined in the previous chapter can route all prefix sequences correctly. This is easily seen by the ranking-based approach, so that we first focus on this one: After the front-end binary sorter, we apply the following mapping before we forward the inputs to the Split modules:

$x_{i,q}$	$x_{i,q+\log(n)}$	$\varrho_\perp(x_{i,q})$	$\varrho_\perp(x_{i,q+\log(n)})$
0	$*$	0	0
1	0	1	0
1	1	1	1

Thus, we define $\varrho_\perp(x_{i,q}) := x_{i,q}$ and $\varrho_\perp(x_{i,q+\log(n)}) := x_{i,q} \wedge x_{i,q+\log(n)}$ which will make sure that bit $x_{i,q+\log(n)}$ is only 1 for valid entries. Ranks are now computed as follows:

$$r_i := \left(\sum_{j=0}^{i} \neg\varrho_\perp(x_{j,q+\log(n)}) \right) - 1 \text{ for } i = 0, \ldots, n-1,$$

i.e., we are computing ranks for those inputs that are either invalid $x_{i,q} = 0$ or have $x_{i,q+\log(n)} = 0$. These inputs are then routed by the configuration we

determined in the previous Section 3.3.2 to the right local target addresses, i.e., their ranks. Since we already have proved that this is always possible for any binary sequence, we conclude the following theorem:

Theorem 4.1 (Ternary Sorting Prefix Sequences) All binary sorters configured by the circuits described in previous chapter can sort any ternary prefix sequence by the ordering $0 \preceq \perp \preceq 1$.

Proof: As we compute the ranks as explained above, inputs with $\mathsf{msb}(x_i) \in \{0, \perp\}$ will be routed correctly to the local addresses given as their ranks. As these are the first contiguous addresses $0, \ldots, k$, and are monotonically increasing, i.e., $i \leq j$ implies $r_i \leq r_j$, it follows that inputs x_i with $\mathsf{msb}(x_i) = 0$ have smaller ranks than inputs x_i with $\mathsf{msb}(x_i) = \perp$. Thus, all $0s$ will appear in a prefix of the sequence before the $\perp$s, and it also follows that inputs x_i with $\mathsf{msb}(x_i) = 1$ are all routed into the suffix. As a result, the output after the ranking-based routing is a sorted ternary sequence $0^i \perp^j 1^k$. Note further that the relative position of inputs x_i with $\mathsf{msb}(x_i) \in \{0, \perp\}$ are maintained while others may be permuted.

As the parity-based configurations also use $\varrho_\perp(x_{j,q+\log(n)})$, we conclude that their configuration is again the same. □

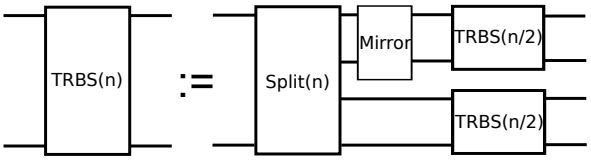


Figure 4.3.: *Ternary RBS network.*

Theorem 4.2 (Routing Partial Permutations) The RBS network with a front-end concentrator as constructed according to Figures 4.1 and 4.3 will correctly route any partial permutation if the circuits described in the previous chapter are used as Split modules.

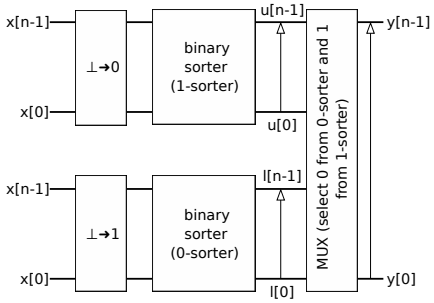
Proof: Finally, a little modification has to be made to the radix-based sorting network so that it can be used as an interconnection network for partial permutations: We therefore implement a TRBS network for n inputs $x_0, \ldots, x_{n-1}$ as shown in Figure 4.3. Due to Figure 4.1, we can assume that the input is a prefix sequence $b^i \perp^j$ with boolean values b. Thus, using one of the three Split modules of the previous section, it will be sorted into a sequence $u_0, \ldots, u_{n-1}$, where sorted means that $\mathsf{msb}(u_i) \preceq \mathsf{msb}(u_{i+1})$ with ordering $0 \preceq \perp \preceq 1$. Since the

inputs $x_0, \ldots, x_{n-1}$ must be a partial permutation of the addresses $0, \ldots, n-1$, there are at most $\frac{n}{2}$ inputs x_i with $\mathsf{msb}(x_i) = 0$ and there are also at most $\frac{n}{2}$ inputs x_i with $\mathsf{msb}(x_i) = 1$. Hence, we conclude that the halves $u_0, \ldots, u_{\frac{n}{2}-1}$ and $u_{\frac{n}{2}}, \ldots, u_{n-1}$ must be of the form $0^m \bot^{\frac{n}{2}-m}$ and $\bot^{\frac{n}{2}-k}1^k$, respectively. For this reason, we reverse the upper half $\bot^{\frac{n}{2}-k}1^k$ by the Mirror module shown in Figure 4.3 so that the inputs to the two TRBS modules become prefix sequences $u_0, \ldots, u_{\frac{n}{2}-1}$ and $u_{n-1}, \ldots, u_{\frac{n}{2}}$. Since both halves are now again prefix sequences, it follows by the induction hypothesis that these are also routed correctly by a TRBS module. □

Hence, the circuits we constructed using the RB-FS-RV network are able to sort all ternary prefix sequences by little modifications: We just have to consider $\varrho_\bot(x_{i,q+\log(n)}) := x_{i,q} \wedge x_{i,q+\log(n)}$ for computing the ranks or the parities to achieve this. Since arbitrary ternary sequences are converted to ternary prefix sequences by the front-end concentrator, and since our Split modules also generate again prefix sequences, these RBS networks can handle all partial permutations.

4.2.2. Constructing Split Modules as Ternary Sorters

We already proved that the RBS network built by Split modules that discussed in previous chapter can route all partial permutations. This is not possible for arbitrary RBS networks using other Split modules. For this reason, we consider ways to generate ternary sorters from binary sorters. Using the ternary sorters as Split modules leads to other RBS networks that can handle partial permutations.



$x_{i,q}$	$x_{i,q+\log(n)}$	$x^0_{i,q}$	$x^0_{i,q+\log(n)}$	$x^1_{i,q}$	$x^1_{i,q+\log(n)}$
0	*	0	1	0	0
1	0	1	0	1	0
1	1	1	1	1	1

$x^0_{i,q} := x_{i,q} \quad x^0_{i,q+\log(n)} := \neg x_{i,q} \vee x_{i,q+\log(n)}$
$x^1_{i,q} := x_{i,q} \quad x^1_{i,q+\log(n)} := x_{i,q} \wedge x_{i,q+\log(n)}$

$l_{i,q}$:	1 ... 1	* ... *	* ... *
$l_{i,q+\log(n)}$:	0 ... 0	1 ... 1	1 ... 1
$u_{i,q}$:	* ... *	* ... *	1 ... 1
$u_{i,q+\log(n)}$:	0 ... 0	0 ... 0	1 ... 1
$y_{i,q}$:	1 ... 1	0 ... 0	1 ... 1
$y_{i,q+\log(n)}$:	0 ... 0	1 ... 1	1 ... 1

Figure 4.4.: *Construction of a Ternary Sorter by two Binary Sorters.*

The left-hand side of Figure 4.4 shows how a ternary sorter can be constructed by two binary sorters that we call the 0-sorter and the 1-sorter, respectively. Both binary sorters obtain the n inputs $x_0, \ldots, x_{n-1}$ after a pre-processing step that modifies the msbs $x_{i,q+\log(n)}$ of the invalid target addresses as shown on the upper right part of Figure 4.4 as $x^0_{i,q+\log(n)}$ and $x^1_{i,q+\log(n)}$ for the 0- and 1-sorter, respectively. Note that after the pre-processing step, only the valid inputs have msbs 0 and 1 for the 0- and 1-sorter, respectively.

After this, the 0-sorter and the 1-sorter sort their input sequences to output sequences $l_0, \ldots, l_{n-1}$ and $u_0, \ldots, u_{n-1}$, respectively, by only considering the modified msbs $x^0_{i,q+\log(n)}$ and $x^1_{i,q+\log(n)}$. Hence, the 0-sorter uses the ordering $0 \prec \{\bot, 1\}$ while the 1-sorter uses ordering $\{0, \bot\} \prec 1$ (regarding the original inputs).

The lower right part of Figure 4.4 shows how the 0- and 1-sorter's output sequences look like in general: The 0-sorter's output sequence starts with values $(l_{i,q}, l_{i,q+\log(n)}) = (1, 0)$, i.e., 0, followed by values $(l_{i,q}, l_{i,q+\log(n)}) = (*, 1)$, i.e., $\bot$ or 1, while the 1-sorters output sequence starts with values $(u_{i,q}, u_{i,q+\log(n)}) = (*, 0)$, i.e., 0 or $\bot$, followed by values $(u_{i,q}, u_{i,q+\log(n)}) = (1, 1)$, i.e., 1. The final stage of multiplexers will then determine output y_i by selecting one of the corresponding values l_i or u_i as follows where l'_i is obtained from l_i by setting its valid bit to 0:

$$y_i := \begin{cases} u_i & : \text{if } u_{i,q} \wedge u_{i,q+\log(n)} \\ l_i & : \text{if } l_{i,q} \wedge \neg l_{i,q+\log(n)} \\ l'_i & : \text{otherwise} \end{cases}$$

Note that the number of valid inputs can be at most n, hence, we never have both $u_{i,q} \wedge u_{i,q+\log(n)}$ and $l_{i,q} \wedge \neg l_{i,q+\log(n)}$. Note further that we have to set $l_{i,q} := 0$ in case l_i is chosen for y_i, but $l_{i,q} \wedge \neg l_{i,q+\log(n)}$ does not hold (this way, we avoid that an input with $(x_{i,q}, x_{i,q+\log(n)}) = (1, 1)$ will be taken from the 0-sorter that has already been copied from the 1-sorter). *It can be easily verified that the circuit shown in* Figure 4.4 *implements a ternary sorter, i.e., any input sequence* $x_0, \ldots, x_{n-1}$ *of values* $\{0, \bot, 1\}$ *is correctly sorted using the total order* $0 \prec \bot \prec 1$.

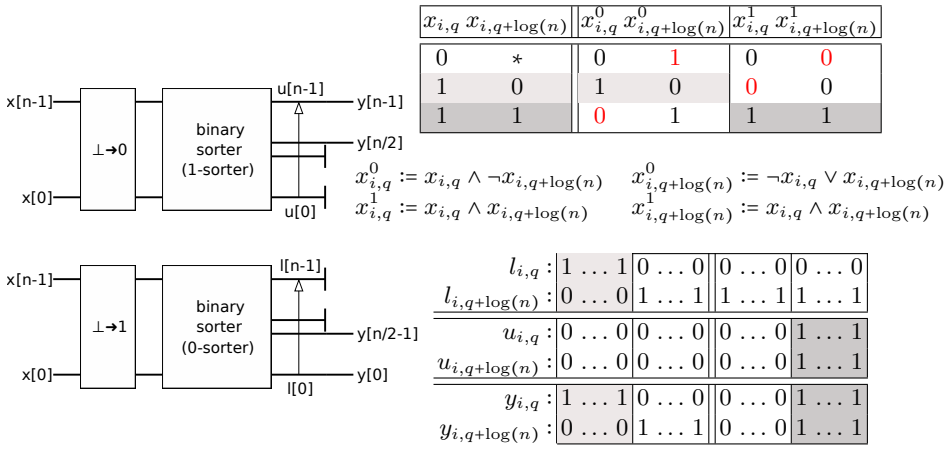
4.2.3. Constructing Split Modules by Ternary Concentrators

We have already mentioned that the Split modules do not have to be ternary sorters to partition the inputs according to their msbs. Instead, it is sufficient to route all inputs x_i with $(x_{i,q}, x_{i,q+\log(n)}) = (1, 1)$ to the upper half and all inputs x_i with $(x_{i,q}, x_{i,q+\log(n)}) = (1, 0)$ to the lower half, while the invalid inputs x_i with $x_{i,q} = 0$ may be routed to any half among the other values routed there.

For this reason, we can also consider the slightly simplified construction given in Figure 4.5. Compared to Figure 4.4, we modify the msbs $x_{i,q+\log(n)}$ of the target addresses in the same way, but additionally invalidate all 1s and 0s in the 0- and 1-sorter, respectively, as shown in the upper right part of Figure 4.5. Hence, the 0-sorter will only have inputs $(x^0_{i,q}, x^0_{i,q+\log(n)}) \in \{(0, 1), (1, 0)\}$, i.e., $\{\bot, 0\}$, and the 1-sorter will only have inputs $(x^1_{i,q}, x^1_{i,q+\log(n)}) \in \{(0, 0), (1, 1)\}$, i.e., $\{\bot, 1\}$.

Again, the 0- and 1-sorter only consider the modified msbs $x^0_{i,q+\log(n)}$ and $x^1_{i,q+\log(n)}$, respectively, to generate their output sequences $l_0, \ldots, l_{n-1}$ and $u_0, \ldots, u_{n-1}$, respectively.

Assuming now that at most $\frac{n}{2}$ inputs x_i satisfy $(x_{i,q}, x_{i,q+\log(n)}) = (1, 0)$ and also at most $\frac{n}{2}$ inputs x_i satisfy $(x_{i,q}, x_{i,q+\log(n)}) = (1, 1)$, we can simply



$x_{i,q}$	$x_{i,q+\log(n)}$	$x^0_{i,q}$	$x^0_{i,q+\log(n)}$	$x^1_{i,q}$	$x^1_{i,q+\log(n)}$
0	∗	0	1	0	0
1	0	1	0	0	0
1	1	0	1	1	1

$x^0_{i,q} := x_{i,q} \wedge \neg x_{i,q+\log(n)}$ $\quad x^0_{i,q+\log(n)} := \neg x_{i,q} \vee x_{i,q+\log(n)}$

$x^1_{i,q} := x_{i,q} \wedge x_{i,q+\log(n)}$ $\quad x^1_{i,q+\log(n)} := x_{i,q} \wedge x_{i,q+\log(n)}$

$l_{i,q}$:	1 … 1	0 … 0	0 … 0	0 … 0
$l_{i,q+\log(n)}$:	0 … 0	1 … 1	1 … 1	1 … 1
$u_{i,q}$:	0 … 0	0 … 0	0 … 0	1 … 1
$u_{i,q+\log(n)}$:	0 … 0	0 … 0	0 … 0	1 … 1
$y_{i,q}$:	1 … 1	0 … 0	0 … 0	1 … 1
$y_{i,q+\log(n)}$:	0 … 0	1 … 1	0 … 0	1 … 1

Figure 4.5.: *Construction of a Ternary Splitter by Binary Sorters.*

determine y_i as follows (see lower right part of Figure 4.5):

$$y_i := \begin{cases} u_i & : \text{if } i \in \{\frac{n}{2}, \ldots, n-1\} \\ l_i & : \text{if } i \in \{0, \ldots, \frac{n}{2}-1\} \end{cases}$$

As long as at most $\frac{n}{2}$ inputs x_i are 0 and 1, the output sequence will even be a sorted ternary sequence. However, if more than $\frac{n}{2}$ inputs x_i should be 0 or more than $\frac{n}{2}$ inputs x_i should be 1, the circuit will omit some of the inputs and will therefore no longer be correct. We therefore do not consider the circuit of Figure 4.5 as a ternary sorter, but each part of it is a $(n, \frac{n}{2})$ concentrator that concentrates on the 0 and 1 values, respectively.

While not yielding a ternary sorter for general ternary sequences, Figure 4.5 still sorts all ternary input sequences that will appear in RBS networks for partial permutations. However, it does not allow some further optimizations as the one shown in the next section.

4.2.4. Constructing Split Modules by Ternary Sorters and Half Cleaners

In section 3.3, we have already shown how (ternary) Split modules with n ports can be constructed using two (ternary) sorters with $\frac{n}{2}$ ports and a half cleaner circuit(is also shown in Figure 4.6). Half cleaners were introduced by Batcher in [Batc68] for the construction of his bitonic sorting networks.

As already proved, it is required to use sorting networks for the construction of Figure 4.6. In particular, the construction shown in the previous section, i.e., in Figure 4.5 cannot be used. Hence, even though our initial construction of Figure 4.4 cannot compete with the one in Figure 4.5, it allows the optimization shown in Figure 4.6.

4.3. Summary

In practice, not all input components have to be always connected to an output component. For this reason, partial permutations have to be implemented by

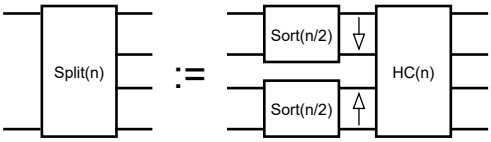


Figure 4.6.: *Construction of a Ternary Splitter by Ternary Sorters and a Half Cleaner.*

non-blocking unicast interconnection networks. The implementation of partial permutations by sorting networks is not straightforward and depends on the used sorting networks and algorithms. For merge-based sorting networks, there is a well-known solution to implement partial permutations, known as the Batcher-Banyan network. However, The same approach, used for merge-based networks to implement the partial permutations, does not work for the radix-based networks.

For RBS network, only one solution exist which is suggested by Narasimha. He claimed that his network can also work with partial permutations if an additional Split module is added on the left side of his RBS network. he did not discuss how that construction could be generalized to other networks.

Therefore, in this chapter, we generalized it to an entire class of configuration circuit. Additionally, We also present three general constructions that can transform binary sorters to ternary Split modules. This way, one can transform any RBS network that has been constructed for total permutations into a one that can work with partial permutations as well.

Chapter 5

Experimental Results and Comparison

Contents

5.1. Asymptotic Complexities 57
5.2. Experimental Results 58
5.2.1. Total Permutations 59
5.2.2. Partial Permutations 59
5.3. Analysis of Results 60

In this chapter, we present experimental results of proposed interconnection networks based on radix-sorting, both for total and partial permutations. We also report asymptotic complexities of our networks and a comparison with other state of the art interconnection networks [Nara94; KoOr90; ChOr94; ChCh96; Batc68] to quantitatively demonstrate the effectiveness of the proposed interconnection networks. Finally, we analyze and discuss the obtained experimental results.

5.1. Asymptotic Complexities

Table 5.1 summarizes the asymptotic complexities obtained for our networks and compares them with others. It shows the comparison of the sizes and the depths of the networks. Note that 'size' denotes here the number of logical gates used in the circuits and 'depth' denotes the length of the longest path through combinational gates from inputs to outputs in the considered circuit. An ideal network has both a size as small as possible and a depth as small as possible. However, there is a trade-off between these two parameters.

In Table 5.1, it can be seen that the crossbar network has the worst size complexity of $O(n^2)$, and that the other networks have the same asymptotic size $O(n(\log(n))^3)$. The depths of the networks differ more: The Narasimha

Table 5.1.: *Comparsion of asymptotic complexities*

network	depth	size
Crossbar	$O(\log(n))$	$O(n^2)$
Narasimha [Nara94]	$O(n)$	$O(n(\log(n))^3)$
Parallel prefix computation	$O(\log(n)^3)$	$O(n(\log(n))^3)$
Ranking based computation	$O(\log(n)^2)$	$O(n(\log(n))^3)$
Batcher-Bitonic [Batc68]	$O(\log(n)^3)$	$O(n(\log(n))^3)$
Chien-Oruç [ChOr94]	$O(\log(n)^3)$	$O(n(\log(n))^3)$
Cheng-Chen [ChCh96]	$O(\log(n)^2)$	$O(n(\log(n))^3)$

network has an unacceptable depth of $O(n)$ while the parallel prefix computation, the Batcher-bitonic, and the Chien-Oruç network have depth $O(\log(n)^3)$, and the ranking-based computation and Cheng-Chen's network even have only depth $O(\log(n)^2)$.

The optimal depth $O(\log(n))$ is only obtained by the crossbar network, but its size of $O(n^2)$ is unacceptable. Thus, we can conclude that the crossbar network cannot be recommended due to the worst size complexity of the network. The size complexity of the other networks is more or less the same, but they differ in their depths.

Depth and size of partial permutation circuits are mainly dominated by the used binary sorters which are also building blocks of total permutation networks. As we have already discussed in Chapter 3, the construction of a split module by a binary sorter and a half cleaner does not improve the asymptotic complexity of the considered binary sorter. Therefore, we can say that the asymptotic complexities of partial permutation circuits are the same as those of the corresponding total permutation networks.

5.2. Experimental Results

The experimental results considered in this chapter were obtained by implementing $Split(n)$ modules and their corresponding RBS networks for 6 binary sorters mentioned in Table 5.1. In the following, we abbreviate these networks by the acronyms of the authors of the paper where these binary sorters were originally published: Nara94 [Nara94], ChOr94 [ChOr94], ChCh96 [ChCh96], Batc68 (the bitonic sorter reduced to one bit) [Batc68], JaSJ17 [JaSJ17], and JaSc18c [JaSc18c]. JaSJ17 and JaSc18c denote our prefix-based and ranking-based circuits, respectively.

All experimental results are made by a netlist generator written in F# and the Cadence® RC compiler (version 14.2) using 65nm CMOS technology. In all tables and plotted graphs, size and depth are the results that were obtained by our netlist generator while the number of cells, chip area, power consumption and latency are the results which are obtained by the Cadence tool.

5.2.1. Total Permutations

Table A.1 and Figure B.1 show the comparison of $Split(n)$ modules for different numbers of inputs n. Similarly, Table A.6 and the plotted graphs in Figure B.6 show the comparison of the corresponding networks for different numbers of inputs n. The runtime for synthesis is several miliseconds up to $n = 64$, however, it exponentially grows for higher values of n. Therefore, we have synthesized all the networks only up to $n = 128$.

Comparing the plotted graphs in Figure B.1, B.6 and tables in Table A.1, A.6, it turns out that size and area of ChOr94 and Batc68 quickly grow with the number of inputs. Area and size of ChCh96, Nara94, and JaSJ17 *Split* modules and networks are more or less the same, but they differ in their depth and latency. The Nara94 network cannot be recommended since its depth and latency grow quickly with number of inputs. Depths of our JaSJ17 and the ChCh96 networks are comparable in the ranges up to $n = 64$, where the former is better for $n \leq 64$, and becomes worse after that. Hence, we would recommend our JaSJ17 network for $n \leq 64$ and the ChCh96n network for $n > 64$. Thus, ChCh96 and our JaSJ17 are better in terms of size, cells, area, depth and latency.

Table A.2, A.7 and the plotted graphs in Figure B.2, B.7 show the experimental results that we have obtained by implementing the *Split* modules and RBS networks using the half cleaner optimization. Wee have already discussed that the half cleaner optimization does not improve the asymptotic complexity of the considered binary sorters. However, after comparing the experimental results of tables Table A.6 and Table A.7, it is clear that it improves the size and depth of the considered binary sorters as well as the network based on these binary sorters.

5.2.2. Partial Permutations

There are different ways to generalize RBS-interconnection networks to deal with partial permutations as discussed in Chapter 4. Depth and size of the partial permutation circuits are mainly dominated by the used binary sorters. While the depth only increases by some constant, the size obviously is twice the size of the binary sorters with few additional gates for mapping the the values and the half cleaner. However, the asymptotic growth of depth and size of circuits are the same as the those for the total permutations.

Table A.3, Table A.5, and Table A.4 show the experimental results obtained for different ternary sorters, ternary concentrators and ternary sorters with half-cleaner constructions, respectively. Similarly, Table A.8, Table A.10 and Table A.9 show the experimental results obtained for the corresponding networks for partial permutations, respectively. The size always improves from Table A.8 via Table A.10 to Table A.9, but the depths are sometimes best for ternary concentrators and sometimes for ternary sorters with half cleaners.

Table A.11 shows the experimental results obtained for front end valid sorter RBS networks. As we have already discussed in Chapter 4, this network construction works for only a special class of binary sorters. Thus, the table

shows only three networks out of the six. Comparing the results of Table A.11 with Table A.8, Table A.10, and Table A.9, it turns out that both size and depth of the front-end valid sorter RBS networks are better. It is also clear that out of these three networks, our JaSJ17 network is overall best.

Table A.12 and the plotted graphs in Figure A.12 show the experimental results for other non-blocking networks and our JasJ17 with front-end valid sorter. Here, we have considered three networks as our competitors: Batcher banyan, Beneš, and a crossbar network. Results for the Beneš network are directly taken from [JiYa17] as they have also realized it using Cadence in 65 nm technology. On comparison, it turns out that the number of cells of Batcher banyan quickly grows with the number of inputs. The number of cells used by the crossbar and our network are more or less the same for the considered inputs. However, they differ in their latency and power. It can be easily be observed that the latency and the power consumption of the crossbar is much better than those of our JaSJ17 network. However, we know that the crossbar network has the worst size complexity of $O(n^2)$, it will grow quickly for larger values of n. Thus, we can conclude that the crossbar network could only be recommended for small value of n, but for larger values of inputs, our JaSJ17 network will be superior.

5.3. Analysis of Results

The experimental results obtained by the netlist generator show the expected growth rates for the size and depth of the networks. We also observed that the of size (obtained by the netlist generator) and the number of cells (obtained by Cadence) is approximately constant for different number of inputs. Similarly, the quotient of depth and latency are also approximately constant for different number of inputs . This brings us to the conclusion that the growth rates of the size and the depth obtained through the two experimental methods are same. However, the absolute numbers obtained through both the methods are not the same for the same number of inputs. The reason behind this is the following: Cadence also optimizes the circuit and reduces the overall gate count. Also, our netlist generator generates an approximate network netlist where it assumes that all components have the same size and depth irrespective of their fan-out, while Cadence generates exact values of these parameters considering the fan-out of all components.

The power dissipation of networks realized in CMOS mainly consist of dynamic and leakage power. The dynamic power is proportional to frequency and parasitic capacitance while leakage power is significantly smaller and can be assumed as zero for all calculations. In order to have a fair comparison, all power dissipation results are obtained at a constant operating frequency of 100 MHz and constant supply voltage of 1.2 V. Therefore, power dissipation should be proportional to effective parasitic capacitance. To a first order approximation, this can be assumed to be proportional to the area. However, as different cells have different area to parasitic capacitance ratio. In particular, the ratio of parasitic capacitance to cell area is larger for components

with larger drive strengths. Therefore, power dissipation is proportional to A^x where x is slightly larger than 1 because for higher values of n there is the large probability that components with higher drive strength will be used. The value of x is found to be in the range of 1.1 to 1.3.

Chapter 6

Conclusions

This thesis presents new radix-based interconnection networks to improve the limitations of existing interconnection networks and reviews and compares these with some existing nonblocking radix-based interconnection networks. Special modifications are required for sorting networks for routing partial permutations. For merge-based sorting networks, there is a well-known solution known as the Batcher-Banyan network. However, for the larger class of RBS networks this does not work, and there is only one solution known by Narasimha's network that can route partial permutations. Thus, this thesis presents a special extension of the proposed networks which allows them to also route partial permutations. Moreover, three general constructions were presented to convert any binary sorter to a ternary split module which is the key to construct a radix-based interconnection network that can cope with partial permutations. The thesis compares also chip designs of these networks with other radix-based sorting networks as well as with the Batcher-Banyan, crossbar and Beneš networks as competitors. As a result, it turns out that the proposed radix-based networks are superior to the Batcher-Banyan and Beneš networks. However, up to a certain value of n, crossbars are better but after that, its size grows quickly beyond reasonable thresholds. Hence, as a result, it turn out that the proposed radix-based networks are superior and could form the basis of large manycore architectures.

Bibliography

[AgIS09] A. Agarwal, C. Iskander, and R. Shankar. "Survey of Network on Chip (NoC) Architectures and Contributions". In: *Journal of Engineering, Computing and Architecture* 3.1 (2009).

[Andr77] S. Andresen. "The Looping Algorithm Extended to Base 2^l Rearrangeable Switching Networks". In: *IEEE Transactions on Communication* 25 (1977), pp. 1057–1063.

[Batc68] K.E. Batcher. "Sorting Networks and their Applications". In: *AFIPS Spring Joint Computer Conference*. Vol. 32. 1968, pp. 307–314.

[BeMi02] L. Benini and G. De Micheli. "Networks on chips: a new SoC paradigm". In: *Computer* 35.1 (Jan. 2002), pp. 70–78.

[BEAC08] Shane Bell, Bruce Edwards, John Amann, Rich Conlin, Kevin Joyce, Vince Leung, John MacKay, Mike Reif, Liewei Bao, John Brown, et al. "Tile64-processor: A 64-core soc with mesh interconnect". In: *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*. IEEE. 2008, pp. 88–598.

[Bene64] V.E. Beneš. "Optimal Rearrangeable Multistage Connecting Networks". In: *The Bell System Technical Journal* 43 (1964), pp. 1641–1656.

[Bene65] V.E. Beneš. *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press, 1965.

[Bene75] V.E. Beneš. "Proving the Rearrangeability of Connecting Networks by Group Calculations". In: *The Bell System Technical Journal* 45 (1975), pp. 421–434.

[BhJS15] A. Bhagyanath, T. Jain, and K. Schneider. "A Time-Predictable Model of Computation". In: *Real-Time Systems Symposium (RTSS)*. Ed. by M. Caccamo. San Antonio, Texas, USA: IEEE Computer Society, 2015, p. 376.

[BjMa06] T. Bjerregaard and S. Mahadevan. "A Survey of Research and Practices of Network-on-Chip". In: *ACM Computing Surveys (CSUR)* 38.1 (Mar. 2006), pp. 1–51.

[BJSH15] H. Bokhari, H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran. "SuperNet: multimode interconnect architecture for manycore chips". In: *Design Automation Conference (DAC)*. San Francisco, CA, USA: ACM, 2015, 85:1–85:6.

[BKMD04] Doug Burger, Stephen W Keckler, Kathryn S McKinley, Mike Dahlin, Lizy K John, Calvin Lin, Charles R Moore, James Burrill, Robert G McDonald, and William Yoder. "Scaling to the End of Silicon with EDGE Architectures". In: *Computer* 37.7 (2004), pp. 44–55.

[Cam03] H. Cam. "Rearrangeability of (2n-1)-Stage Shuffle-Exchange Networks". In: *SIAM Journal of Control and Optimization (SICON)* 32.3 (2003), pp. 557–585.

[ChCh96] W.-J. Cheng and W.-T. Chen. "A New Self-Routing Permutation Network". In: *IEEE Transactions on Computers* 45.5 (May 1996), pp. 630–636.

[CaFo99] H. Çam and J.A.B. Fortes. "Work-Efficient Routing Algorithms for Rearrangeable Symmetrical Networks". In: *IEEE Transactions on Parallel and Distributed Systems* 10.7 (July 1999), pp. 733–741.

[Chun78] F.R.K. Chung. "On Concentrators, Superconcentrators, Generalizers, and Nonblocking Networks". In: *The Bell Systems Technical Journal* 58.8 (Oct. 1978), pp. 1765–1777.

[Clos53] C. Clos. "A Study of Non-Blocking Switching Networks". In: *Bell System Technical Journal* 32.2 (1953), pp. 406–424.

[ChOr94] M.V. Chien and A.Y. Oruç. "High Performance Concentrators and Superconcentrators Using Multiplexing Schemes". In: *IEEE Transactions on Communications* 42.11 (Nov. 1994), pp. 3045–3050.

[Corp94] Henk Corporaal. "Design of transport triggered architectures". In: *VLSI, 1994. Design Automation of High Performance VLSI Systems. GLSV'94, Proceedings., Fourth Great Lakes Symposium on*. IEEE. 1994, pp. 130–135.

[DaTo04] W.J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[Feng81] T. Feng. "A Survey of Interconnection Networks". In: *IEEE Computer* 14.12 (Dec. 1981), pp. 12–27.

[FeSe94] T.-Y. Feng and S.-W. Seo. "A New Routing Algorithm for a Class of Rearrangeable Networks". In: *IEEE Transactions on Computers (T-C)* 43.11 (1994), pp. 1270–1280.

[GoLi98] L.R. Goke and G. Jack Lipovski. "Banyan Networks for Partitioning Multiprocessor Systems". In: *25 Years of the International Symposia on Computer Architecture (ISCA)*. Barcelona, Spain: ACM, 1998, pp. 117–124.

[GaPa83] Z. Galil and W.J. Paul. “An Efficient General-Purpose Parallel Computer”. In: *Journal of the ACM (JACM)* 30.2 (1983), pp. 360–387.

[HuKn84] A. Huang and S. Knauer. “Starlite: A wideband digital switch”. In: *Global Telecommunications Conference (GLOBECOM)*. 1984, pp. 121–125.

[Hols09] R. Holsmark. “Deadlock Free Routing in Mesh Networks on Chip with Regions”. PhD. PhD thesis. Linköping Studies in Science and Technology, 2009.

[HoSR98] E. Horowitz, S. Sahni, and S. Rajasekaran. *Computer Algorithms*. Computer Science Press, 1998.

[HeWC04] J. Henkel, W. Wolf, and S. Chakradhar. “On-chip networks: a scalable, communication-centric embedded system design paradigm”. In: *International Conference on VLSI Design*. IEEE Computer Society, 2004, pp. 845–851.

[JaOr93] C.Y. Jan and A.Y. Oruç. “Fast Self-Routing Permutation Switching on an Asymptotically Minimum Cost Network”. In: *IEEE Transactions on Computers* 42.12 (Dec. 1993), pp. 1469–1479.

[JaSc16] T. Jain and K. Schneider. “Verifying the Concentration Property of Permutation Networks by BDDs”. In: *Formal Methods and Models for Codesign (MEMOCODE)*. Ed. by E. Leonard and K. Schneider. Kanpur, India: IEEE Computer Society, 2016, pp. 43–53.

[JaSc18] T. Jain and K. Schneider. “The Half Cleaner Lemma: Constructing Efficient Interconnection Networks from Sorting Networks”. In: *Parallel Processing Letters* 28.1 (Mar. 2018).

[JaSc18a] T. Jain and K. Schneider. “Routing Partial Permutations in General Interconnection Networks based on Radix Sorting”. In: *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*. Ed. by O. Bringmann and A. von Bernuth. ISBN 978-3-00-059317-8. Tübingen, Germany, 2018.

[JaSc18c] T. Jain and K. Schneider. “Routing Partial Permutations in Interconnection Networks based on Radix Sorting”. In: *2018 13th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE. 2018, pp. 1–10.

[JaSJ17] T. Jain, K. Schneider, and A. Jain. “An Efficient Self-Routing and Non-Blocking Interconnection Network on Chip”. In: *Network on Chip Architectures (NoCArc)*. Ed. by M. Ebrahimi and T. Hollstein. Boston, MA, USA: ACM, 2017, 4:1–4:6.

[JaSJ17b] T. Jain, K. Schneider, and A. Jain. "Deriving Concentrators from Binary Sorters Using Half Cleaners". In: *Reconfigurable Computing and FPGAs (ReConFig)*. Ed. by P. Athanas, R. Cumplido, C. Feregrino, and R. Sass. Cancun, Mexico: IEEE Computer Society, 2017, pp. 1–6.

[JaSJ17c] T. Jain, K. Schneider, and A. Jain. *Deriving Concentrators from Binary Sorters Using Half Cleaners*. Poster Presented at ReConFig. Dec. 2017.

[JiYa14] Y. Jiang and M. Yang. "On circuit design of on-chip non-blocking interconnection networks". In: *System-on-Chip Conference (SOCC)*. Las Vegas, NV, USA: IEEE Computer Society, 2014, pp. 192–197.

[JiYa17] Yikun Jiang and Mei Yang. "Hardware design of parallel switch setting algorithm for Benes networks". In: *International Journal of High Performance Systems Architecture* 7.1 (2017), pp. 26–40.

[Kann05] R. Kannan. "The KR-Beneš Network: A Control-Optimal Rearrangeable Permutation Network". In: *IEEE Transactions on Computers (T-C)* 54.5 (2005), pp. 534–544.

[KiDA07] J. Kim, W.J. Dally, and D. Abts. "Flattened butterfly: a cost-efficient topology for high-radix networks". In: *International Symposium on Computer Architecture (ISCA)*. Ed. by D.M. Tullsen and B. Calder. San Diego, California, USA: ACM, 2007, pp. 126–137.

[KeME16] S. Kerrison, D. May, and K. Eder. "A Beneš Based NoC Switching Architecture for Mixed Criticality Embedded Systems". In: *Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*. Lyon, France: IEEE Computer Society, 2016, pp. 125–132.

[KoOr90] D.M. Koppelman and A.Y. Oruç. "A Self-Routing Permutation Network". In: *Journal of Parallel and Distributed Computing* 10.2 (1990), pp. 140–151.

[KrSn83] C.P. Kruskal and M. Snir. "The Performance of Multistage Interconnection Networks for Multiprocessors". In: *IEEE Transactions on Computers* 32.12 (Dec. 1983), pp. 1091–1098.

[KrSn86] C.P. Kruskal and M. Snir. "A Unified Theory of Interconnection Network Structure". In: *Theoretical Computer Science (TCS)* 48 (1986), pp. 75–94.

[KiYM97] M.K. Kim, H. Yoon, and S.R. Maeng. "On the correctness of inside-out routing algorithm". In: *IEEE Transactions on Computers (T-C)* 46.7 (July 1997), pp. 820–823.

[Lai00] Wei Kuang Lai. "Performing permutations on interconnection networks by regularly changing switch states". In: *IEEE Transactions on Parallel & Distributed Systems* 8 (2000), pp. 829–837.

[Lawr75] D.H. Lawrie. “Access and Alignment of Data in an Array Processor”. In: *IEEE Transactions on Computers (T-C)* 24 (Dec. 1975), pp. 1145–1155.

[LBFS98] Walter Lee, Rajeev Barua, Matthew Frank, Devabhaktuni Srikrishna, Jonathan Babb, Vivek Sarkar, and Saman Amarasinghe. “Space-time scheduling of instruction-level parallelism on a raw machine”. In: *ACM SIGPLAN Notices*. Vol. 33. 11. ACM. 1998, pp. 46–57.

[Lee85] K.Y. Lee. “On the Rearrangeability of (2 log(N)-1) Stage Permutation Networks”. In: *IEEE Transactions on Computers (T-C)* 34.5 (1985), pp. 412–425.

[Leis85a] C.E. Leiserson. “Fat-trees: Universal Networks for Hardware Efficient Supercomputing”. In: *IEEE Transactions on Computers (T-C)* 34.10 (Oct. 1985), pp. 892–901.

[LeLi96] T.T. Lee and S.Y. Liew. “Parallel routing algorithms in Beneš-Clos networks”. In: *Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation.* San Francisco, CA, USA: IEEE Computer Society, 1996, pp. 279–286.

[LeOr95a] C.-Y. Lee and A.Y. Oruç. “Design of Efficient and Easily Routable Generalized Connectors”. In: *IEEE Transactions on Communications* 43.2-4 (1995), pp. 646–650.

[LuZh02] E. Lu and S.Q. Zheng. “A Fast Parallel Routing Algorithm for Beneš Group Switches”. In: *International Parallel and Distributed Processing Symposium (IPDPS)*. Cambridge, Massachusetts, USA, 2002, pp. 67–72.

[Mein03] J.D. Meindl. “Beyond Moore’s Law”. In: *Computing in Science and Engineering (CiSE)* 5.1 (2003), pp. 20–24.

[MaGN79] G.M. Masson, G.C. Gingher, and S. Nakamura. “A Sampler of Circuit Switching Networks”. In: *IEEE Computer* 12.6 (1979), pp. 32–48.

[Nara88] M.J. Narasimha. “The Batcher-Banyan self-routing network: universality and simplification”. In: *IEEE Transactions on Communications* 36.10 (Oct. 1988), pp. 1175–1178.

[Nara94] M.J. Narasimha. “A Recursive Concentrator Structure with Applications to Self-Routing Switching Networks”. In: *IEEE Transactions on Communications* 42.2-4 (1994), pp. 896–898.

[NgDu01] H.Q. Ngo and D.-Z. Du. “Remarks on Beneš Conjecture”. In: *Switching Networks: Recent Advances*. Ed. by D.-Z. Du and H.Q. Ngo. Kluwer, 2001. Chap. Remarks on Beneš Conjecture, pp. 257–258.

[NaSa81] D. Nassimi and S. Sahni. “A Self-Routing Beneš Network and Parallel Permutation Algorithms”. In: *IEEE Transactions on Computers (T-C)* 30.5 (1981), pp. 332–340.

[NaSa82] D. Nassimi and S. Sahni. “Parallel Algorithms to Set Up the Beneš Permutation Network”. In: *IEEE Transactions on Computers (T-C)* 31.2 (Feb. 1982), pp. 148–154.

[OrOr85] A.Y. Oruç and M.Y. Oruç. “Equivalence Relations Among Interconnection Networks”. In: *Journal of Parallel and Distributed Computing* 2 (1985), pp. 30–49.

[PaAv12] K.V. Palem and L. Avinash. “What to do about the end of Moore’s law, probably!” In: *Design Automation Conference (DAC)*. Ed. by P. Groeneveld, D. Sciuto, and S. Hassoun. San Francisco, California, USA: ACM, 2012, pp. 924–929.

[Pate81] J.H. Patel. “Performance of Processor-Memory Interconnections for Multiprocessors”. In: *IEEE Transactions on Computers* 30.10 (Oct. 1981), pp. 771–780.

[Pins73] M.S. Pinsker. “On the Complexity of a Concentrator”. In: *International Teletraffic Conference (ITC)*. Stockholm, Sweden, 1973, 318:1–318:4.

[Pipp77] N. Pippenger. “Superconcentrators”. In: *SIAM Journal on Computing* 6.2 (June 1977), pp. 298–304.

[Pipp78a] N. Pippenger. “On Rearrangeable and Non-Blocking Switching Networks”. In: *Journal of Computer and System Sciencies (JCSS)* 17.2 (Oct. 1978), pp. 145–162.

[RJMC95] David F Robinson, Dan Judd, Philip K McKinley, and Betty HC Cheng. “Efficient multicast in all-port wormhole-routed hypercubes”. In: *Journal of parallel and distributed computing* 31.2 (1995), pp. 126–140.

[RaVa87] C.S. Raghavendra and A. Varma. “Rearrangebility of the five stage shuffle-exchange network for N=8”. In: *IEEE Transactions on Communication* 35.8 (Aug. 1987), pp. 808–812.

[SDMS12] K. Sewell, R.G. Dreslinski, T. Manville, S. Satpathy, et al. “Swizzle-Switch Networks for Many-Core Systems”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 2.2 (June 2012), pp. 278–294.

[SeFL99] S.-W. Seo, T.-Y. Feng, and H.-I. Lee. “Permutation Realizability and Fault Tolerance Property of the Inside-Out Routing Algorithm”. In: *IEEE Transactions on Parallel and Distributed Systems* 10.9 (1999), pp. 946–957.

[ScJu96] T. Schwederski and M. Jurczyk. *Verbindungsnetze – Strukturen und Eigenschaften*. Springer, 1996.

[ScRe39] F.J. Scudder and J.N. Reynolds. “Crossbar Dial Telephone Switching System”. In: *Bell System Technical Journal* 18.2 (Jan. 1939), pp. 76–118.

[SSMP07] Steven Swanson, Andrew Schwerin, Martha Mercaldi, Andrew Petersen, Andrew Putnam, Ken Michelson, Mark Oskin, and Susan J Eggers. "The wavescalar architecture". In: *ACM Transactions on Computer Systems (TOCS)* 25.2 (2007), p. 4.

[ThCh10] A. Thamarakuzhi and J.A. Chandy. "2-Dilated Flattened Butterfly: A nonblocking switching network". In: *International Conference on High Performance Switching and Routing*. Richardson, Texas, USA: IEEE Computer Society, 2010, pp. 153–158.

[TSBS07] Martin Thuresson, Magnus Själander, Magnus Björk, Lars Svensson, Per Larsson-Edefors, and Per Stenstrom. "FlexCore: Utilizing exposed datapath control for efficient computing". In: *Journal of Signal Processing Systems* 57.1 (2009), pp. 5–19.

[TuMe03] J.S. Turner and R. Melen. "Multirate Clos Networks". In: *IEEE Communications Magazine* 41.10 (Oct. 2003), pp. 38–44.

[WSCH15] Luc Waeijen, Dongrui She, Henk Corporaal, and Yifan He. "A low-energy wide SIMD architecture with explicit datapath". In: *Journal of Signal Processing Systems* 80.1 (2015), pp. 65–86.

[Waks69] A. Waksman. "A Permutation Network". In: *Journal of the ACM (JACM)* 15.1 (Jan. 1969), pp. 159–163.

[WuFe80] C. Wu and T. Feng. "On a Class of Multistage Interconnection Networks". In: *IEEE Transactions on Computers (T-C)* 29.8 (July 1980), pp. 694–702.

Appendix A

Experimental Tables

Table A.1.: *CNC-BIN-HC0*

Size						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	18	32	8	8	8	23
4	156	139	183	82	82	164
8	816	493	1068	364	363	669
16	3360	1541	4277	1260	1256	2209
32	12000	4441	14310	3844	3833	6524
64	38976	12101	43047	10852	10826	17986
128	118272	31649	120712	29060	29003	47315
Number of Cells						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	9	13	8	7	7	16
4	124	114	155	65	65	118
8	692	430	988	299	298	567
16	2904	1366	4097	1059	1055	1908
32	10448	3954	13934	3275	3264	5675
64	34048	10782	42283	9323	9297	15693
128	103488	28186	119176	25099	25042	41342
Area (μm^2)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	51	76	44	41	41	85
4	714	682	859	382	382	673
8	4006	2569	5472	1760	1754	3314
16	16859	8139	22679	6234	6208	11197
32	60764	23504	77105	19276	19207	33346
64	198266	63968	233925	54865	54701	92239
128	603190	166965	659228	147679	147319	242992
Depth						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	3	5	1	1	1	6
4	9	9	5	4	4	10
8	18	12	12	7	8	15
16	30	15	22	13	12	21
32	45	18	35	22	18	27
64	63	21	51	39	26	33
128	84	24	70	72	34	39
Latency (pico seconds)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	468.2000	907.5000	401.4000	405.4000	405.4000	746.5000
4	961.7000	1525.9000	929.4000	748.9000	710.6000	1185.5000
8	1742.2000	2089.9000	1783.8000	1154.8000	1235.8000	1978.0000
16	2836.6000	2663.7000	2985.7000	1769.8000	1824.1000	2759.4000
32	4240.3000	3248.9000	4522.7000	2698.4000	2688.2000	3543.4000
64	5965.4000	3834.0000	6353.2000	4419.2000	3667.2000	4348.6000
128	7233.6000	4422.0000	8667.5000	7802.3000	4797.0000	5132.3000
Power (mW)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	0.8322	1.9526	0.5814	0.5645	0.5645	1.2575
4	13.2451	18.6319	16.0588	8.4931	8.4931	14.8418
8	78.1262	99.8236	147.8342	59.0447	58.5408	105.0164
16	399.7474	502.5331	904.7243	370.9658	367.4707	570.7982
32	1546.2277	2121.1031	3768.1479	1794.4202	1777.3061	2588.2974
64	5862.5201	9025.8116	15363.6389	7652.9236	7590.2861	11567.3425
128	19569.1842	33783.4438	50089.4197	29526.6717	29309.2850	42773.6437

Table A.2.: *CNC-BIN-HC1*

Size						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	18	18	18	18	18	18
4	104	148	100	100	100	130
8	544	542	686	396	396	560
16	2352	1706	3120	1320	1318	1930
32	8800	4906	11282	3960	3952	5858
64	29696	13298	35668	11080	11058	16440
128	92928	34570	103446	29512	29460	43780
Number of Cells						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	16	16	16	16	16	16
4	78	86	80	74	74	92
8	452	432	642	320	320	426
16	2016	1436	3024	1104	1102	1640
32	7632	4220	11074	3368	3360	5066
64	25888	11556	35220	9512	9490	14312
128	81216	30204	102486	25480	25428	38220
Area (μm^2)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	92	92	92	92	92	92
4	449	499	449	430	430	518
8	2616	2553	3577	1870	1870	2452
16	11703	8499	16802	6471	6458	9580
32	44385	24973	61446	19766	19715	29693
64	150747	68340	195265	55860	55721	83999
128	473372	178488	567892	149685	149357	224433
Depth						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	3	3	3	3	3	3
4	6	8	4	4	4	9
8	12	12	8	7	7	13
16	21	15	15	10	11	18
32	33	18	25	16	15	24
64	48	21	38	25	21	30
128	66	24	54	42	29	36
Latency (pico seconds)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	502.4000	502.4000	502.4000	502.4000	502.4000	502.4000
4	722.4000	1157.7000	647.4000	646.9000	646.9000	991.5000
8	1244.0000	1779.4000	1179.2000	1002.1000	954.0000	1439.8000
16	2059.9000	2349.4000	2074.3000	1430.3000	1522.7000	2244.3000
32	3176.3000	2930.3000	3305.0000	2056.0000	2126.3000	3011.4000
64	4604.9000	3518.9000	4744.9000	2963.2000	2922.2000	3803.6000
128	5743.8000	4079.2000	6720.6000	4660.9000	3946.5000	4574.3000
Power (mW)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	1.3456	1.3456	1.3456	1.3456	1.3456	1.3456
4	7.8871	10.2075	7.2164	7.1151	7.1151	9.1829
8	46.8118	62.7319	66.1365	40.5886	40.5886	58.4196
16	256.1821	316.2490	485.6127	225.0268	224.1208	308.3723
32	1030.1926	1326.9856	2301.3149	1006.8466	1001.5025	1383.6883
64	4084.6397	6021.9044	10189.5893	4936.3471	4899.8487	6729.3180
128	14033.7302	23879.5470	36147.3203	19813.1422	19674.7495	27033.6676

Table A.3.: *CNC-TRP-HC0*

Size						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	96	124	76	76	76	106
4	576	478	710	364	364	528
8	2576	1546	3416	1288	1286	1898
16	9696	4522	12442	3960	3952	5858
32	32640	12402	39196	11208	11186	16568
64	101632	32522	112990	30024	29972	44292
128	298752	82498	306848	77320	77206	113830
Number of Cells						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	74	82	73	67	67	88
4	488	412	654	311	311	420
8	2232	1372	3256	1107	1105	1646
16	8464	4044	12082	3427	3419	5128
32	28576	11108	38444	9747	9725	14550
64	89088	29116	111462	26195	26143	38938
128	262016	73780	303776	67603	67489	100092
Area (μm^2)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	417	468	401	382	382	487
4	2806	2414	3614	1798	1798	2395
8	12916	8069	18009	6436	6423	9561
16	49137	23810	66830	19984	19933	29927
32	166216	65408	212636	56938	56799	85092
64	518869	171410	616460	153189	152861	227953
128	1527489	434225	1679993	395638	394918	586281
Depth						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	8	10	6	6	6	11
4	14	14	10	9	9	15
8	23	17	17	12	13	20
16	35	20	27	18	17	26
32	50	23	40	27	23	32
64	68	26	56	44	31	38
128	89	29	75	77	39	44
Latency (pico seconds)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	739.6000	1178.9000	677.4000	676.8000	676.8000	1017.9000
4	1276.9000	1801.9000	1215.9000	1041.8000	986.6000	1472.4000
8	2109.3000	2372.5000	2126.4000	1467.5000	1549.8000	2277.5000
16	3241.8000	2950.1000	3228.6000	2055.0000	2103.5000	3041.3000
32	4281.7000	3538.1000	4820.0000	2980.7000	2937.5000	3832.9000
64	6446.2000	4098.8000	6829.3000	4712.8000	4031.9000	4604.0000
128	7727.8000	4721.8000	9211.7000	8081.9000	4952.2000	5427.5000
Power (mW)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	6.3763	9.1667	6.5712	6.2745	6.2745	7.6234
4	57.2612	72.1800	81.9911	45.2470	45.2786	62.8794
8	255.5837	272.3994	468.2553	211.9002	210.4990	300.9913
16	1108.9580	1373.8542	2285.0621	1020.2022	1013.4040	1383.5394
32	4330.1079	5808.7771	9037.7924	4853.5155	4827.8073	6565.3476
64	15126.1619	23647.6484	32266.2432	20190.5643	20079.1430	27431.4282
128	47867.1009	84740.5279	103167.2178	75012.1582	74664.0011	100933.7498

Table A.4.: *CNC-TRP-HC1*

Size						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	28	28	28	28	28	28
4	328	384	288	288	288	348
8	1584	1324	1932	1096	1096	1424
16	6464	4020	8480	3504	3500	4724
32	23168	11284	29572	10160	10144	13956
64	75648	30052	90696	27664	27620	38384
128	230656	77076	256588	72080	71976	100616

Number of Cells						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	25	25	25	25	25	25
4	258	274	260	244	244	286
8	1336	1128	1772	926	926	1144
16	5576	3520	8056	2990	2986	4068
32	20160	9976	28628	8742	8726	12144
64	66080	26664	88712	23942	23898	33548
128	201856	68472	252508	62630	62526	88116

Area (μm^2)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	145	145	145	145	145	145
4	1466	1567	1447	1397	1397	1605
8	7697	6584	9832	5352	5352	6547
16	32303	20638	44650	17371	17346	23621
32	117130	58590	158570	50938	50837	70824
64	384602	156700	491179	139758	139480	196067
128	1176251	402472	1397708	366030	365373	515558

Depth						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	4	4	4	4	4	4
4	12	14	10	10	10	15
8	18	18	14	13	13	19
16	27	21	21	16	17	24
32	39	24	31	22	21	30
64	54	27	44	31	27	36
128	72	30	60	48	35	42

Latency (pico seconds)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	553.4000	553.4000	553.4000	553.4000	553.4000	553.4000
4	1026.9000	1466.2000	967.9000	964.1000	964.1000	1305.2000
8	1585.0000	2095.8000	1515.5000	1341.8000	1280.5000	1773.4000
16	2436.9000	2669.6000	2350.4000	1746.9000	1831.2000	2580.7000
32	3330.8000	3253.3000	3577.3000	2365.7000	2426.9000	3355.3000
64	5045.5000	3817.5000	5163.0000	3282.3000	3276.1000	4119.5000
128	6209.6000	4410.2000	7202.3000	4981.8000	4174.1000	4895.8000

Power (mW)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	1.5244	1.5244	1.5244	1.5244	1.5244	1.5244
4	28.3911	33.4193	27.3574	26.5222	26.5222	30.6783
8	145.5789	153.4029	195.1405	117.9520	117.9203	150.4810
16	669.7401	767.1961	1192.8364	553.3545	551.6426	736.4296
32	2795.8870	3200.2776	5347.3195	2718.5154	2705.8285	3607.1063
64	10183.1689	13750.3972	21252.3494	11608.4982	11548.4263	15302.8558
128	33774.7034	51934.3679	73025.3970	48238.6083	48007.6255	63151.8275

Table A.5.: *CNC-TRC-HC0*

Size						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	75	104	56	56	56	84
4	518	422	654	308	308	468
8	2428	1402	3272	1144	1142	1746
16	9336	4170	12090	3608	3600	5490
32	31792	11570	38364	10376	10354	15704
64	99680	30602	111070	28104	28052	42308
128	294336	78146	302496	72968	72854	109350
Number of Cells						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	39	48	24	36	36	52
4	272	300	372	195	195	297
8	1354	1060	2218	763	761	1291
16	5536	3244	8920	2511	2503	4193
32	19816	9156	29814	7455	7433	12223
64	64640	24508	89460	20687	20635	33363
128	197024	63156	250162	54735	54621	87093
Area (μm^2)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	218	272	126	202	202	284
4	1561	1769	2047	1128	1128	1697
8	7829	6275	12256	4452	4439	7535
16	32126	19210	49320	14701	14651	24578
32	115234	54186	164867	43718	43579	71756
64	376413	144921	494705	121417	121089	195963
128	1148448	373158	1383348	321402	320682	511624
Depth						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	5	7	4	4	4	8
4	11	11	8	6	6	12
8	20	14	15	9	10	17
16	32	17	25	15	14	23
32	47	20	38	24	20	29
64	65	23	54	41	28	35
128	86	26	73	74	36	41
Latency (pico seconds)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	574.4000	1004.5000	478.3000	497.5000	497.5000	852.7000
4	1086.2000	1616.7000	989.9000	847.5000	801.4000	1291.5000
8	1892.3000	2176.9000	1854.8000	1255.6000	1341.0000	2088.9000
16	3001.6000	2750.7000	3095.5000	1866.7000	1937.5000	2873.7000
32	4424.1000	3333.2000	4574.0000	2768.6000	2761.2000	3658.7000
64	6037.4000	3917.3000	6450.1000	4505.5000	3775.4000	4429.4000
128	8235.3000	4475.4000	8822.2000	7872.7000	4931.4000	5214.8000
Power (mW)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	3.0463	5.7740	1.9925	3.0134	3.0134	4.5019
4	32.4649	54.8658	43.6811	28.0092	28.0408	45.0029
8	155.7463	214.8474	306.1900	143.6000	142.1988	228.9463
16	754.8696	1116.9435	1686.8566	758.9662	752.1680	1110.8239
32	3040.2113	4801.8868	7066.6719	3735.4532	3709.7451	5409.2782
64	11053.3428	19878.2120	26351.0494	16082.2308	15970.8095	23133.0407
128	36140.0703	72217.4487	86547.9604	61202.4591	60854.3020	86640.4504

Table A.6.: *NET-BIN-HC0*

Size						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	18	32	8	8	8	23
4	192	203	199	98	98	210
8	1200	899	1466	560	559	1089
16	5760	3339	7209	2380	2374	4387
32	23520	11119	28728	8604	8581	15298
64	86016	34339	100503	28060	27988	48582
128	290304	100327	321718	85180	84979	144479
Number of Cells						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	9	13	8	7	7	16
4	142	152	155	72	72	152
8	976	734	1298	436	435	895
16	4856	2834	6693	1924	1918	3698
32	20160	9622	27320	7116	7093	13071
64	74368	30026	96923	23548	23476	41835
128	252224	88238	313022	72188	71987	125012
Area (μm^2)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	51	76	44	41	41	85
4	815	910	859	423	423	856
8	5637	4389	7191	2566	2559	5178
16	28132	16916	37061	11324	11286	21554
32	117029	57336	151227	41883	41737	76454
64	432323	178639	536379	138589	138134	245147
128	1467837	524243	1731986	424816	423546	733287
Depth						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	3	5	1	1	1	6
4	12	14	6	5	5	16
8	30	26	18	12	13	31
16	60	41	40	25	25	52
32	105	59	75	47	43	79
64	168	80	126	86	69	112
128	252	104	196	158	103	151
Latency (pico seconds)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	468.2000	907.5000	401.4000	405.4000	405.4000	746.5000
4	1153.2000	2184.3000	1040.1000	877.6000	839.3000	1665.4000
8	2618.7000	3993.3000	2478.6000	1755.7000	1798.4000	3400.0000
16	5178.6000	6380.3000	5104.7000	3248.8000	3345.8000	5882.7000
32	9142.2000	9352.5000	9189.9000	5670.5000	5757.3000	9149.4000
64	14830.9000	12909.8000	15069.4000	9813.0000	9147.8000	13221.3000
128	21787.8000	17055.1000	23304.1000	17338.6000	13668.1000	18076.9000
Power (mW)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	0.8322	1.9526	0.5814	0.5645	0.5645	1.2575
4	13.5515	28.5858	17.0525	10.3574	10.3574	24.0663
8	149.6018	210.7147	230.8509	98.2670	97.8582	192.6794
16	1058.1856	1441.0819	2119.5930	810.4535	806.1196	1454.6979
32	5969.9147	7547.1088	13084.9849	4663.8750	4627.4714	8334.4215
64	30557.6707	35702.3586	71587.0034	24720.2680	24533.4596	40874.0635
128	146049.0382	143112.1068	345672.4540	104879.3045	104183.6489	174555.0044

Table A.7.: *NET-BIN-HC1*

Size						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	18	18	18	18	18	18
4	140	184	136	136	136	166
8	824	910	958	668	668	892
16	4000	3526	5036	2656	2654	3714
32	16800	11958	21354	9272	9260	13286
64	63296	37214	78376	29624	29578	43012
128	219520	108998	260198	88760	88616	129804

Number of Cells						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	16	16	16	16	16	16
4	110	118	112	106	106	124
8	672	692	866	532	532	678
16	3360	2820	4756	2168	2166	3044
32	14352	9860	20586	7704	7692	11154
64	54592	31276	76392	24920	24874	36620
128	190400	92756	255270	75320	75176	111460

Area (μm^2)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	92	92	92	92	92	92
4	632	682	632	613	613	701
8	3880	4069	4840	3096	3096	3880
16	19463	16638	26483	12663	12651	17643
32	83310	58249	114413	45093	45017	64979
64	317367	184838	424090	146046	145755	213956
128	1108107	548164	1416072	441777	440867	652346

Depth						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	3	3	3	3	3	3
4	9	11	7	7	7	12
8	21	23	15	14	14	25
16	42	38	30	24	25	43
32	75	56	55	40	40	67
64	123	77	93	65	61	97
128	189	101	147	107	90	133

Latency (pico seconds)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	502.4000	502.4000	502.4000	502.4000	502.4000	502.4000
4	948.1000	1383.4000	873.1000	872.6000	872.6000	1217.2000
8	1915.4000	2913.7000	1750.2000	1598.0000	1549.9000	2390.4000
16	3698.6000	4982.2000	3464.9000	2751.6000	2795.9000	4380.4000
32	6598.2000	7635.8000	6376.9000	4530.9000	4645.5000	7115.1000
64	10926.4000	10878.0000	10656.4000	7217.4000	7291.0000	10642.0000
128	16393.5000	14680.5000	16954.8000	11601.6000	10960.8000	14939.6000

Power (mW)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	1.3456	1.3456	1.3456	1.3456	1.3456	1.3456
4	13.5706	15.8721	12.8811	12.7798	12.7798	14.5749
8	93.1995	123.0287	114.3379	85.8074	85.7757	112.1950
16	638.5895	879.7108	1018.9073	591.2659	590.5817	832.3628
32	3650.9391	5198.3616	7229.8854	3347.1387	3336.6214	4936.1671
64	20203.3967	27302.4211	43245.1110	17544.9031	17469.2485	26635.3669
128	97858.1025	118981.4100	223260.4063	78912.1013	78540.4893	121561.5283

Table A.8.: *NET-TRP-HC0*

Size						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	96	124	76	76	76	106
4	768	726	862	516	516	740
8	4112	2998	5140	2320	2318	3378
16	17920	10518	22722	8600	8588	12614
32	68480	33438	84640	28408	28362	41796
64	238592	99398	282270	86840	86696	127884
128	775936	281294	871388	251000	250598	369598
Number of Cells						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	74	82	73	67	67	88
4	636	600	800	445	445	600
8	3504	2572	4856	1997	1995	2894
16	15472	9188	21794	7421	7409	10916
32	59520	29484	82032	24589	24543	36382
64	208128	88084	275526	75373	75229	111702
128	196156	249948	234338	218349	217947	323496
Area (μm^2)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	417	468	401	382	382	487
4	3640	3501	4417	2562	2562	3393
8	20196	15071	26843	11561	11548	16651
16	89528	53952	120517	43105	43030	63228
32	345272	173312	453669	143148	142858	211549
64	1209414	518035	1523799	439486	438576	651051
128	1813982	1470295	2351176	1274610	1272070	1888382
Depth						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	8	10	6	6	6	11
4	22	24	16	15	15	26
8	45	41	33	27	28	46
16	80	61	60	45	45	72
32	130	84	100	72	68	104
64	198	110	156	116	99	142
128	287	139	231	193	138	186
Latency (pico seconds)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	739.6000	1178.9000	677.4000	676.8000	676.8000	1017.9000
4	1739.8000	2731.7000	1616.6000	1441.9000	1386.7000	2223.7000
8	3572.4000	4823.3000	3391.5000	2632.7000	2659.8000	4246.9000
16	6537.5000	7496.7000	6245.0000	4411.0000	4486.6000	7011.5000
32	10542.5000	10758.1000	10650.5000	7115.0000	7147.4000	10567.7000
64	16712.0000	14580.2000	17067.5000	11551.1000	10902.6000	14895.0000
128	18176.6000	19025.3000	22479.9000	19356.3000	15578.1000	20045.8000
Power (mW)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	6.3763	9.1667	6.5712	6.2745	6.2745	7.6234
4	81.9568	110.0755	106.8742	70.7551	70.7551	102.8880
8	481.1277	585.0780	752.7020	411.5899	410.8098	585.0680
16	3029.6446	3544.3889	5310.1564	2412.9998	2404.4858	3469.2340
32	15561.6242	17441.8435	29052.3419	13063.8332	13000.8675	19148.1143
64	71702.9282	78710.6211	146082.2742	59580.8489	59292.9244	87049.1599
128	92041.5869	311112.8757	181913.2227	242820.6518	241712.7122	352159.6294

Table A.9.: *NET-TRP-HC1*

Size						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	28	28	28	28	28	28
4	384	440	344	344	344	404
8	2352	2204	2620	1784	1784	2232
16	11168	8428	13720	7072	7068	9188
32	45504	28140	57012	24304	24280	32332
64	166656	86332	204720	76272	76180	103048
128	563968	249740	666028	224624	224336	306712
Number of Cells						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	25	25	25	25	25	25
4	308	324	310	294	294	336
8	1952	1824	2392	1514	1514	1824
16	9480	7168	12840	6018	6014	7812
32	39120	24312	54308	20778	20754	27768
64	144320	75288	197328	65498	65406	89084
128	140274	219048	169249	193626	193338	266284
Area (μm^2)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	145	145	145	145	145	145
4	1757	1858	1738	1687	1687	1896
8	11210	10603	13308	8727	8727	10389
16	54723	41845	71266	34824	34799	45004
32	226575	142280	301102	120586	120435	160832
64	837753	441259	1093383	380931	380349	517732
128	1295590	1284989	1677563	1127892	1126072	1551021
Depth						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	4	4	4	4	4	4
4	16	18	14	14	14	19
8	34	36	28	27	27	38
16	61	57	49	43	44	62
32	100	81	80	65	65	92
64	154	108	124	96	92	128
128	226	138	184	144	127	170
Latency (pico seconds)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	553.4000	553.4000	553.4000	553.4000	553.4000	553.4000
4	1303.6000	1742.9000	1244.6000	1240.8000	1240.8000	1581.9000
8	2611.9000	3599.7000	2483.4000	2305.9000	2244.6000	3088.7000
16	4772.1000	5992.6000	4482.3000	3776.1000	3799.1000	5408.0000
32	7826.2000	8969.2000	7710.5000	5865.1000	5949.3000	8486.6000
64	12595.0000	12510.0000	12489.6000	8870.7000	8948.7000	12329.4000
128	13551.0000	16643.5000	16397.6000	13575.8000	12846.1000	16948.5000
Power (mW)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	1.5244	1.5244	1.5244	1.5244	1.5244	1.5244
4	37.8530	42.2299	36.1680	35.3328	35.3328	39.2845
8	271.5421	316.7599	330.2248	243.7332	243.7015	290.9532
16	1473.0293	1736.6309	2036.7871	1267.4096	1266.1503	1667.2239
32	8337.7871	9344.5826	13862.1410	7154.9615	7136.3007	9684.5555
64	40900.1835	46086.5194	75671.9110	35590.0280	35477.0894	48255.8320
128	53673.4678	198472.2801	101129.6243	157510.1224	156942.3425	215807.5654

Table A.10.: *NET-TRC-HC0*

Size						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	75	104	56	56	56	84
4	668	630	766	420	420	636
8	3764	2662	4804	1984	1982	3018
16	16864	9494	21698	7576	7564	11526
32	65520	30558	81760	25528	25482	38756
64	230720	91718	274590	79160	79016	119820
128	755776	261582	851676	231288	230886	348990
Number of Cells						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	39	48	24	36	36	52
4	350	420	420	267	267	405
8	2054	1900	3058	1297	1295	2149
16	9644	7044	15036	5105	5093	8491
32	39104	23244	59886	17665	17619	29205
64	142848	70996	209232	56017	55873	91773
128	144542	205148	188820	166769	166367	270639
Area (μm^2)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	218	272	126	202	202	284
4	1997	2464	2300	1532	1532	2291
8	11823	11204	16856	7516	7504	12420
16	55772	41617	83032	29734	29658	49418
32	226778	137420	330931	103187	102896	170592
64	829968	419761	1156568	327791	326881	537147
128	1325659	1212681	1873156	976984	974443	1585918
Depth						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	5	7	4	4	4	8
4	16	18	12	10	10	20
8	36	32	27	19	20	37
16	68	49	52	34	34	60
32	115	69	90	58	54	89
64	180	92	144	99	82	124
128	266	118	217	173	118	165
Latency (pico seconds)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	574.4000	1004.5000	478.3000	497.5000	497.5000	852.7000
4	1383.9000	2372.1000	1185.8000	1068.3000	1022.2000	1877.6000
8	2999.5000	4268.1000	2686.5000	2047.2000	2086.5000	3722.0000
16	5724.4000	6742.1000	5356.8000	3637.2000	3747.3000	6319.0000
32	9871.8000	9798.6000	9490.8000	6129.1000	6231.8000	9701.0000
64	15632.5000	13439.2000	15402.3000	10357.9000	9730.5000	13853.7000
128	16789.5000	17637.9000	20355.7000	17953.9000	14385.2000	18791.8000
Power (mW)						
n	Batc68	ChCh96	ChOr94	Nara94	JaSJ17	JaSc18c
2	3.0463	5.7740	1.9925	3.0134	3.0134	4.5019
4	43.4914	79.2644	51.3901	40.4376	40.4376	68.1163
8	268.4023	423.7202	448.2497	259.8300	259.1449	417.1688
16	1876.5566	2680.6319	3690.7993	1678.0244	1669.9539	2604.9523
32	10426.3503	13740.2453	22062.7514	9788.0757	9725.3160	15109.6888
64	50482.0207	63721.1678	117156.6877	46392.2651	46110.9553	70448.2731
128	70832.1232	257641.8359	156864.5754	195164.0131	194060.4007	292581.2895

Table A.11.: *TNT-BIN-HC0*

Size			
n	Nara94	JaSJ17	JaSc18c
2	42	42	72
4	300	300	494
8	1332	1330	2166
16	4856	4846	7812
32	15808	15774	25182
64	47744	47646	75400
128	136640	136382	214194
Number of Cells			
n	Nara94	JaSJ17	JaSc18c
2	36	36	55
4	242	242	369
8	1080	1078	1787
16	3984	3974	6559
32	13112	13078	21363
64	39960	39862	64401
128	115192	114934	183819
Area (μm^2)			
n	Nara94	JaSJ17	JaSc18c
2	212	212	306
4	1425	1425	2114
8	6360	6347	10411
16	23459	23396	38372
32	77197	76982	125228
64	235223	234604	377888
128	677960	676330	1079162
Depth			
n	Nara94	JaSJ17	JaSc18c
2	3	3	13
4	10	10	27
8	20	22	47
16	39	38	74
32	70	62	107
64	126	96	146
128	231	138	191
Latency (pico seconds)			
n	Nara94	JaSJ17	JaSc18c
2	640.8000	640.8000	1326.5000
4	1483.2000	1387.0000	2700.0000
8	2811.3000	2937.9000	5252.7000
16	4948.1000	5120.1000	8520.5000
32	8268.7000	8287.5000	12601.5000
64	14120.7000	12732.4000	17432.7000
128	25093.5000	18527.5000	23099.7000
Power (mW)			
n	Nara94	JaSJ17	JaSc18c
2	3.6670	3.6670	5.7177
4	43.7881	43.6931	68.7773
8	246.0875	244.8548	414.3183
16	1402.0442	1386.9506	2298.2443
32	6987.0256	6904.0813	11788.2386
64	29786.1777	29425.9866	49250.2605
128	113772.3410	112437.6853	189943.3253

Table A.12.: *Comparison With Other Networks*

Number of Cells				
n	BatcherBanyan	JaSJ17(front-end valid sorter)	CrossbarPartial	Beneš
4	462	242	112	100
8	2196	1078	488	1810
16	8512	3974	2016	8110
32	29184	13078	7776	36200
64	91392	39862	30464	132000
128	271552	114934	119936	—
Latency (pico seconds)				
n	BatcherBanyan	JaSJ17(front-end valid sorter)	CrossbarPartial	Beneš
4	2475.2000	1387.0000	662.9000	103.0000
8	4836.7000	2937.9000	753.0000	834.0000
16	7638.5000	5120.1000	874.4000	2300.0000
32	10669.9000	8287.5000	964.5000	3680.0000
64	15162.4000	12732.4000	1059.4000	5590.0000
128	21157.7000	18527.5000	1160.2000	—
Power (mW)				
n	BatcherBanyan	JaSJ17(front-end valid sorter)	CrossbarPartial	Beneš
2	6.2566	3.6670	1.8360	
4	71.5909	43.6931	8.8459	
8	451.0856	244.8548	34.1927	
16	2483.0007	1386.9506	117.7731	
32	12083.4982	6904.0813	350.3599	
64	50469.1407	29425.9866	977.6127	
128	195253.6889	112437.6853	2716.8357	

Appendix B

Experimental Graphs

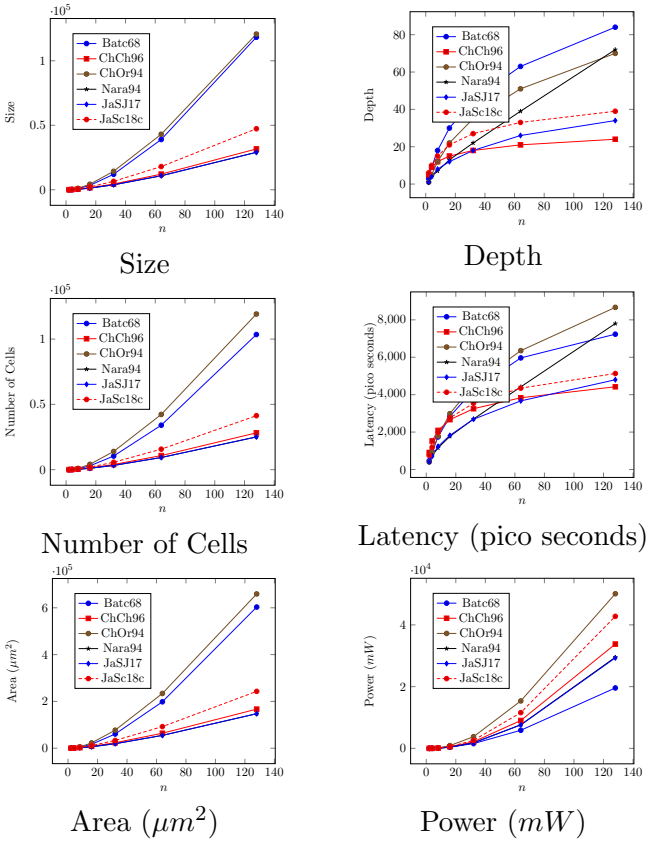


Figure B.1.: *CNC-BIN-HC0*

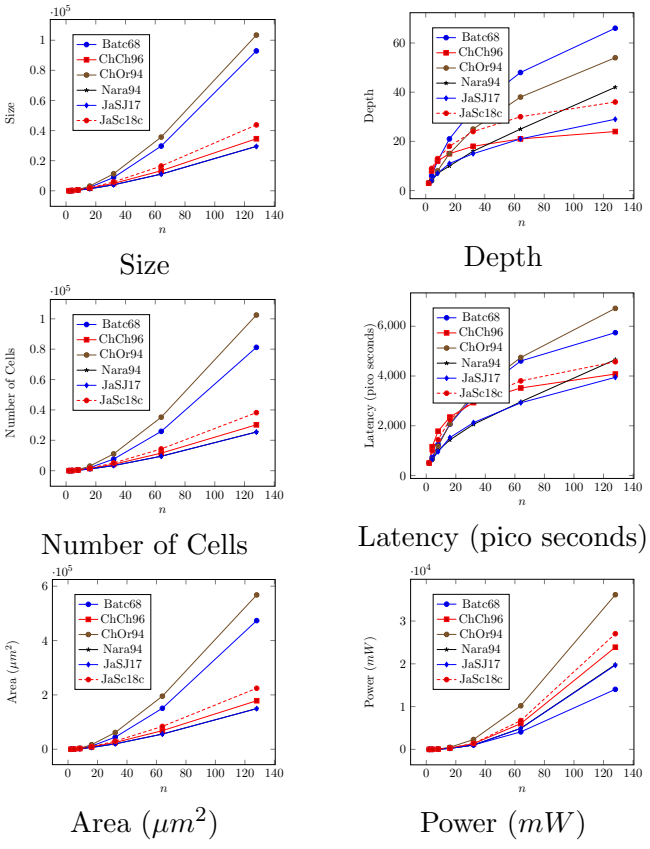


Figure B.2.: *CNC-BIN-HC1*

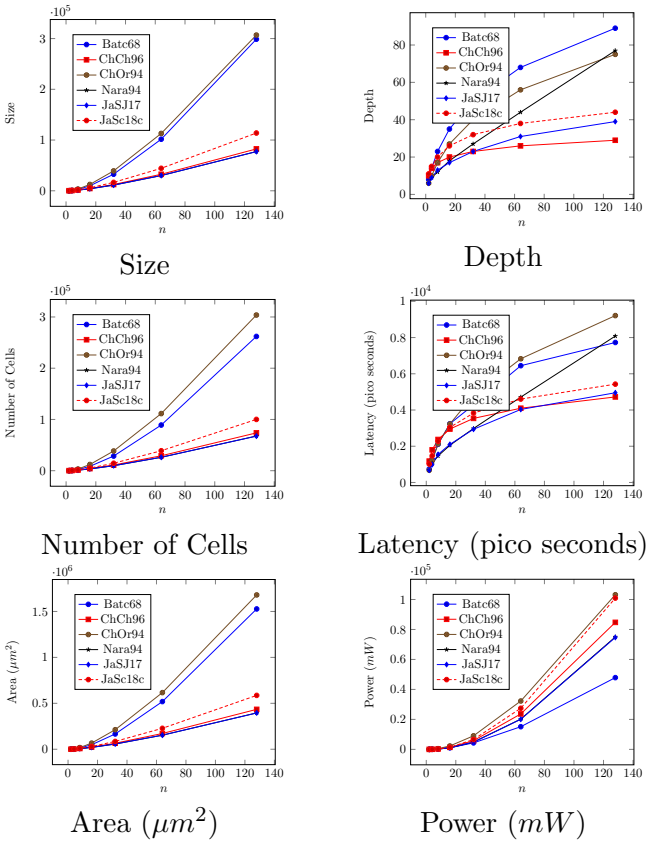


Figure B.3.: *CNC-TRP-HC0*

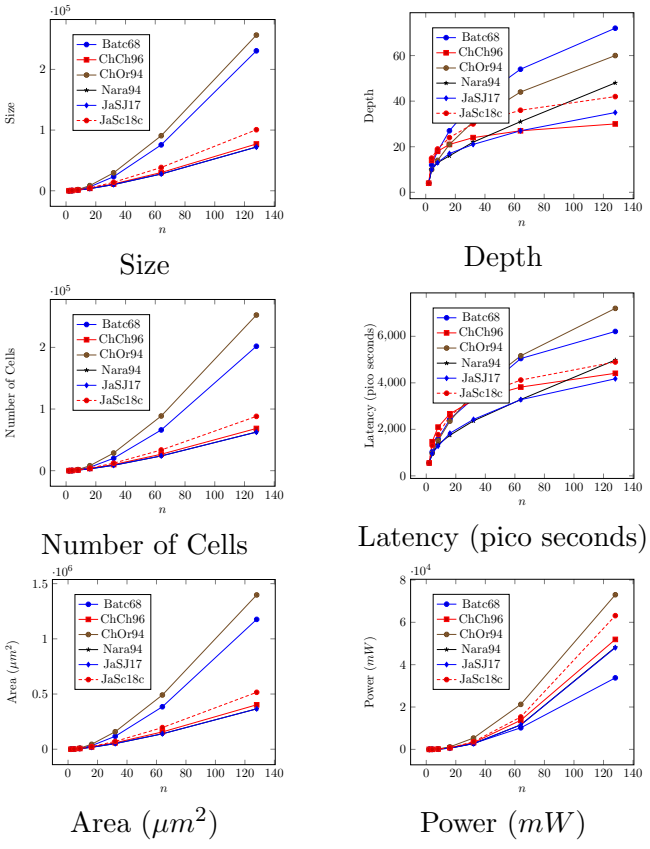


Figure B.4.: *CNC-TRP-HC1*

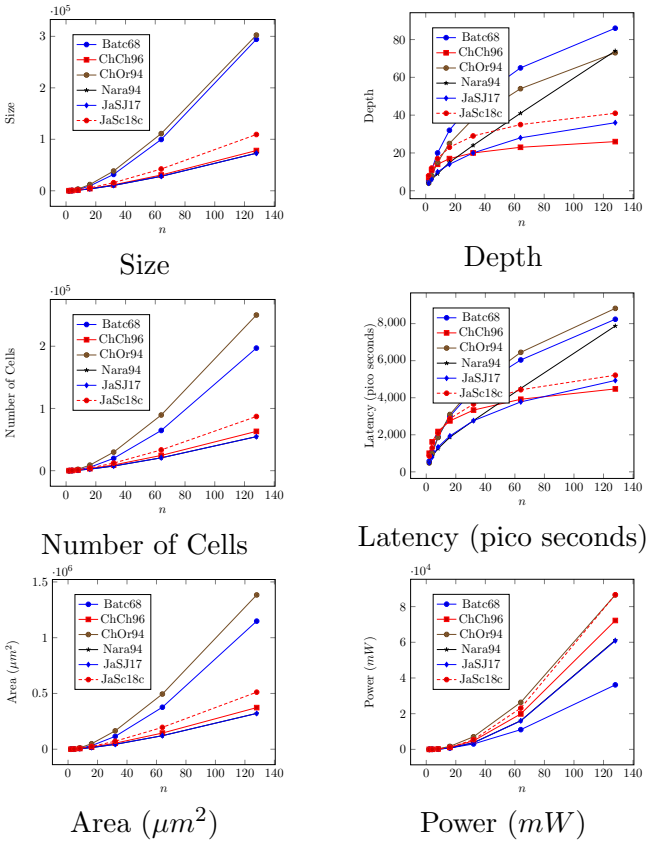


Figure B.5.: *CNC-TRC-HC0*

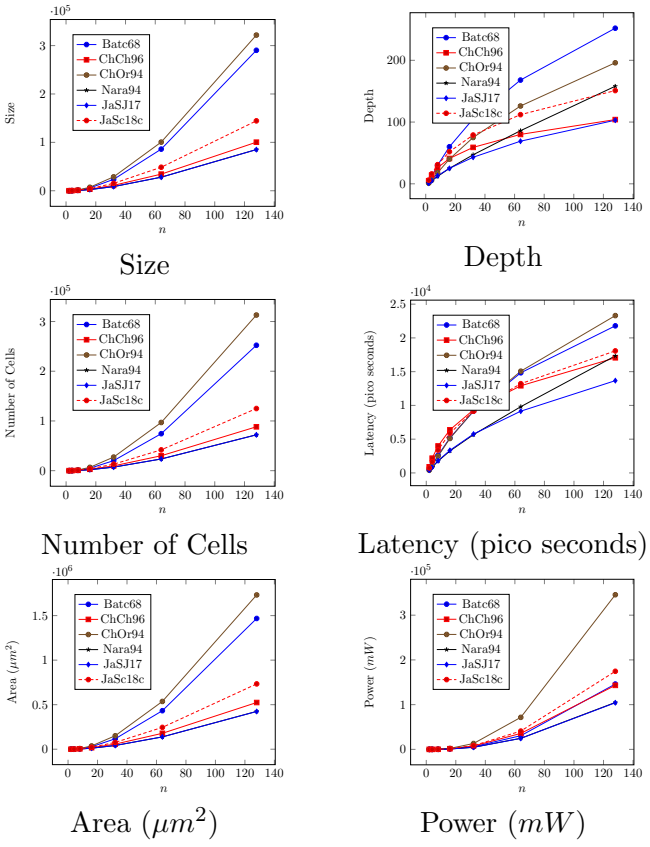


Figure B.6.: *NET-BIN-HC0*

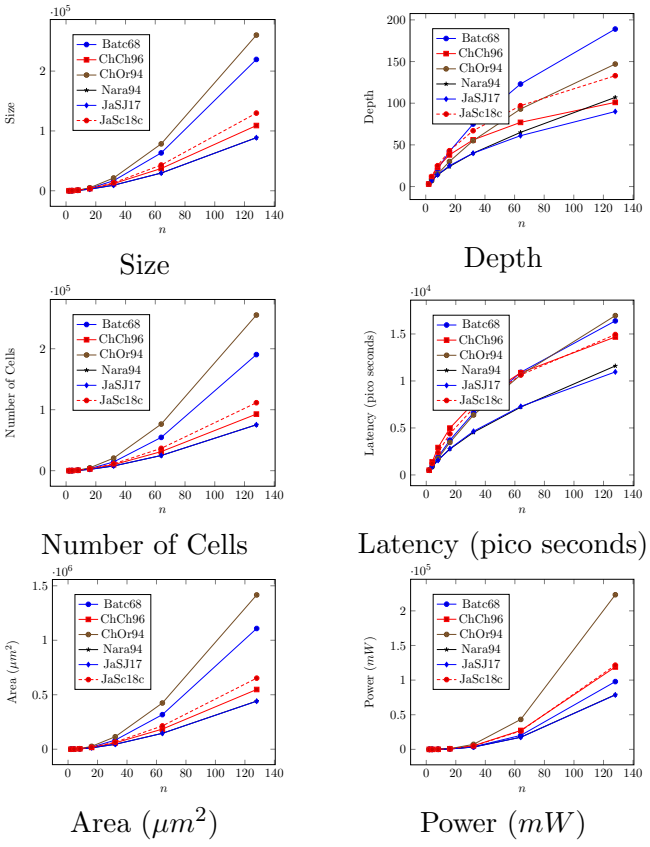


Figure B.7.: *NET-BIN-HC1*

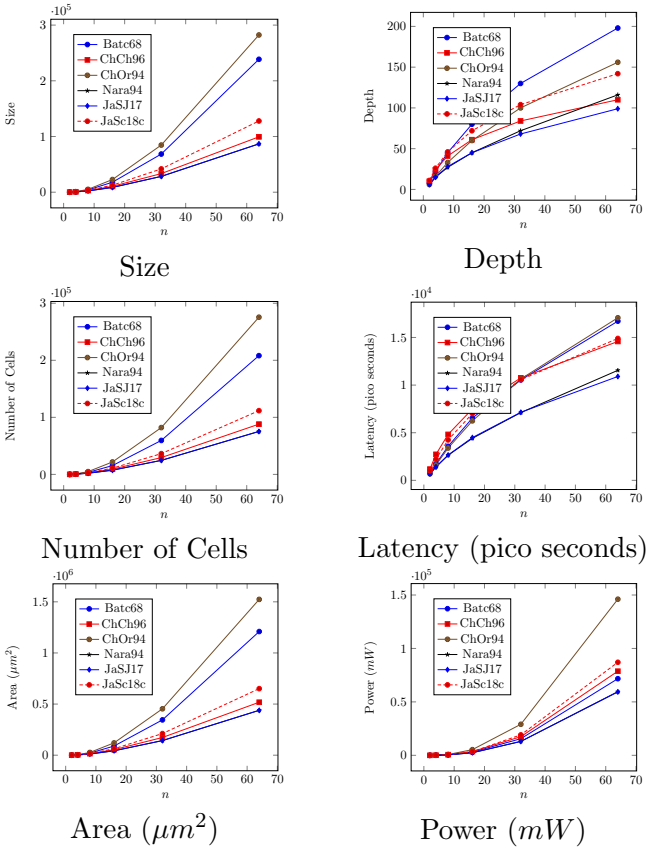


Figure B.8.: *NET-TRP-HC0*

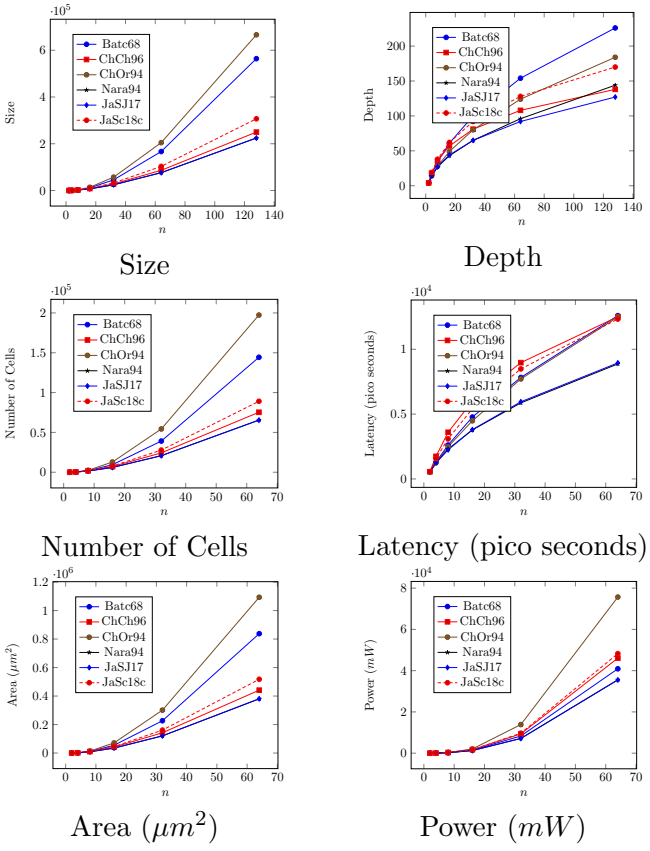


Figure B.9.: *NET-TRP-HC1*

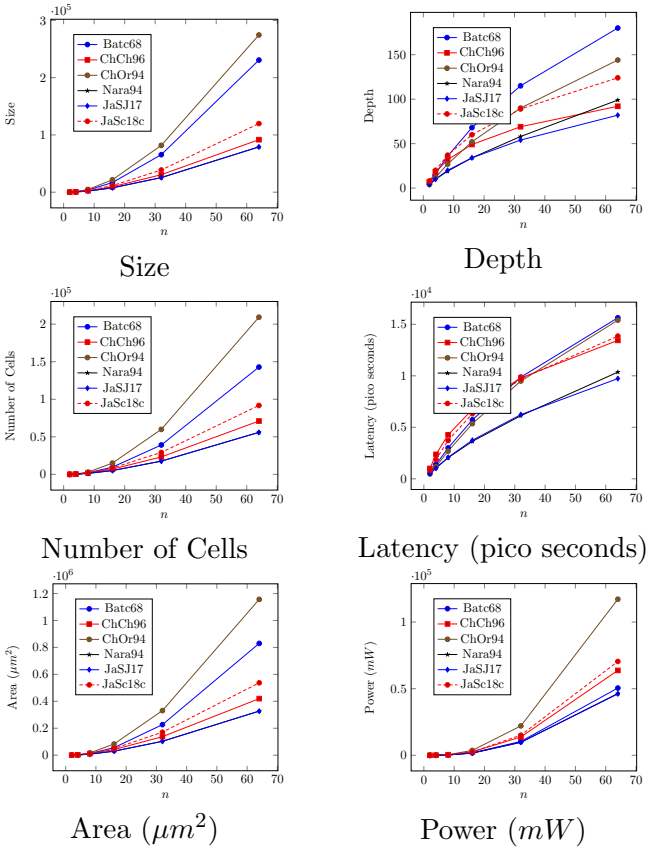


Figure B.10.: *NET-TRC-HC0*

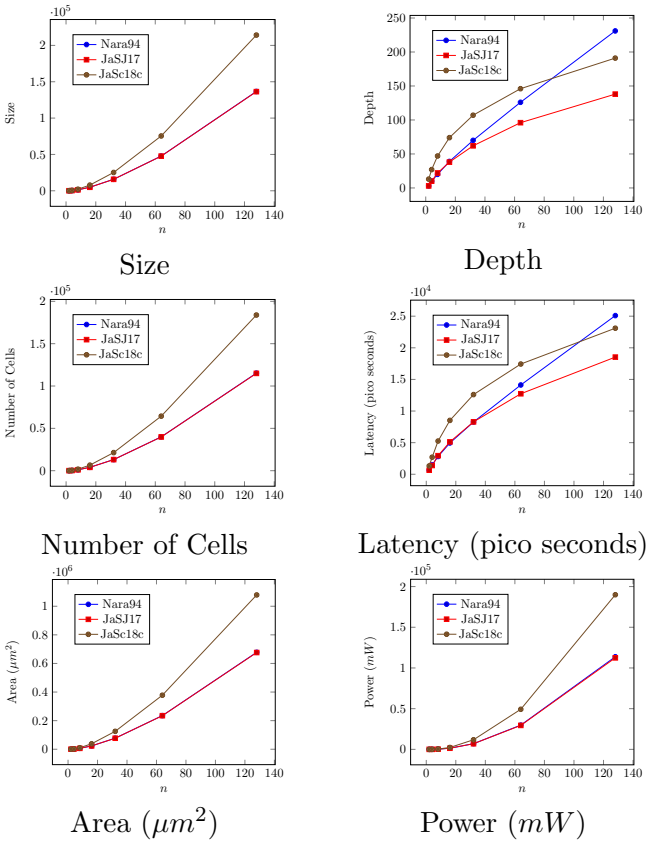


Figure B.11.: *TNT-BIN-HC0*

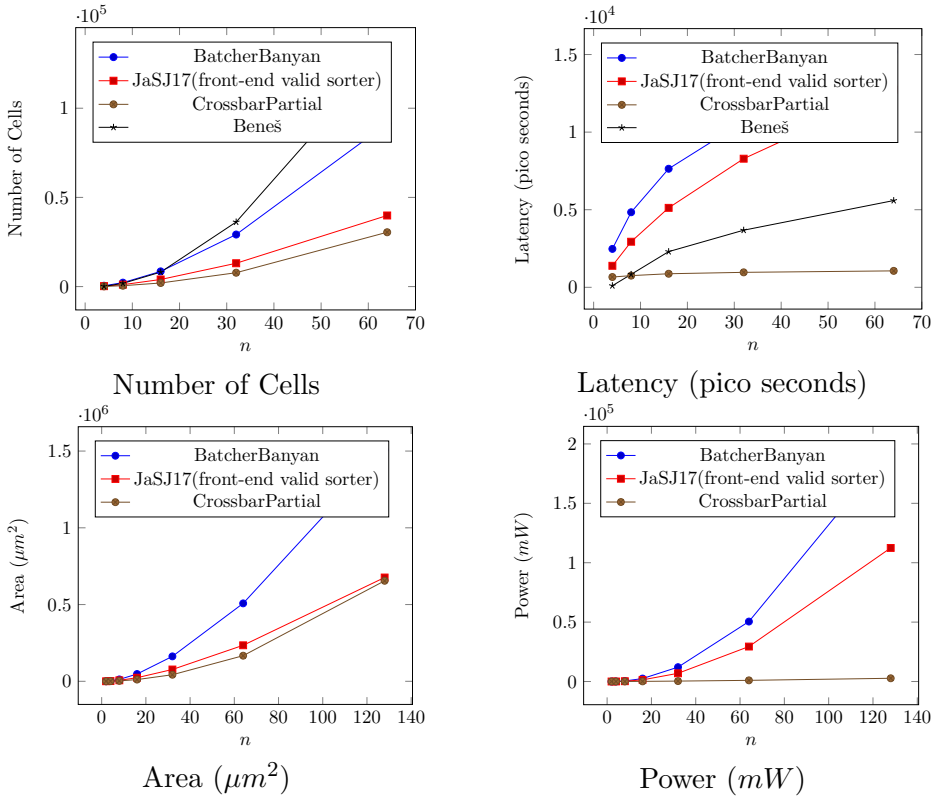


Figure B.12.: *Comparison with Others Networks*

Appendix C

Curriculum Vitae

Professional Experience

2015–2018	**Wissenschaftlicher Hilfskraft** TU Kaiserslautern, Germany Fachbereich Informatik, Arbeitsgruppe Matritzen
2013–2014	**Research Associate** IIT Bombay, India Electrical Department, PI: Prof. Virendra Singh
2012–2013	**Project Trainee** IIT Bombay, India Electrical Department, PI: Prof. Maryam S. Bhagini
2011–2012	**SMDP-II, Lecturer** SGSITS Indore, India Electronics and Instrumentation Department
2010–2010	**Lecturer** SVIT Indore, India Electrical and Electronics Department
2009–2010	**Lecturer** IIST Indore, India Electronics and Communication Department
2004–2007	**Lecturer** SVITS Indore, India Electrical Engineering Department

Education

2007–2009	**M.Tech in Microelctronics and VLSI Design** SGSITS Indore Thesis: Reconfigurable Hardware Implementation of Median Filter for Image Processing
1999–2003	**B.E. in Electrical Engineering** SGSITS Indore Thesis: Implementation of continuation Power flow

Schulausbildung

1986-1998	**High School** SKBVN Indore, India Majors: Mathematics, Physics, Chemistry